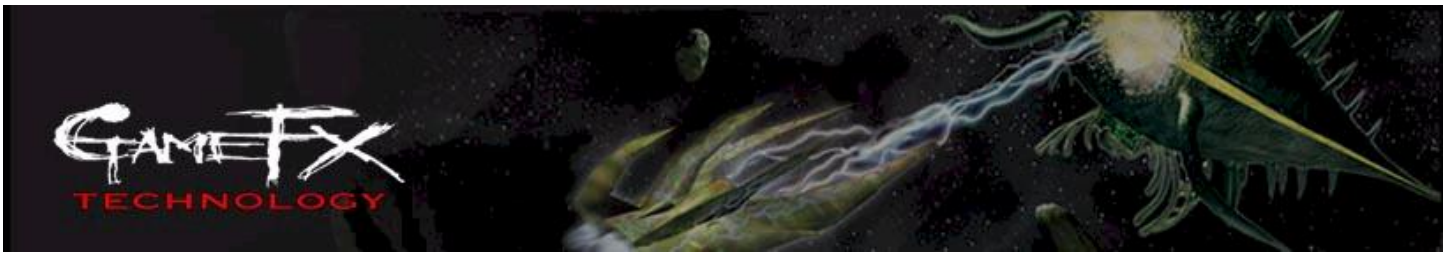
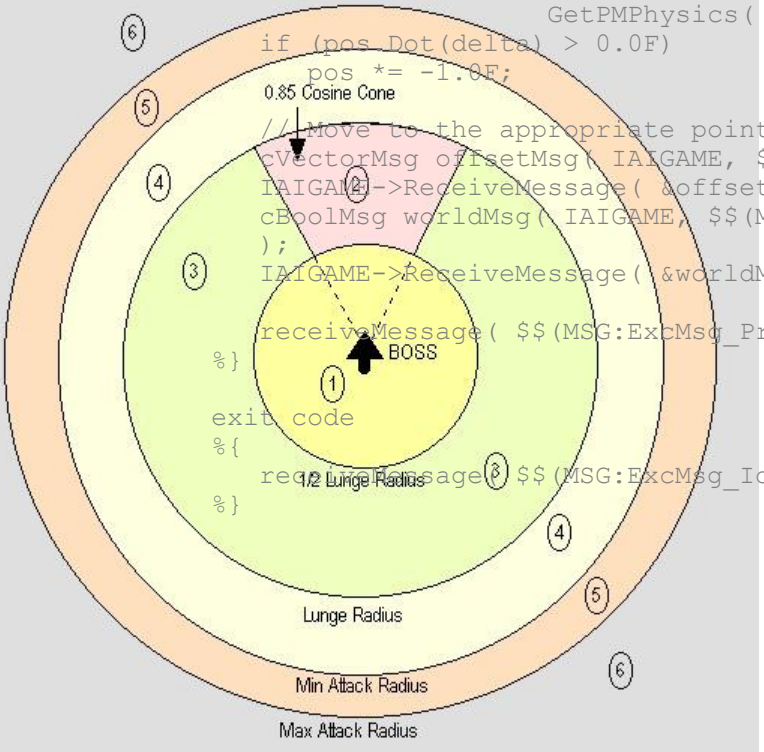
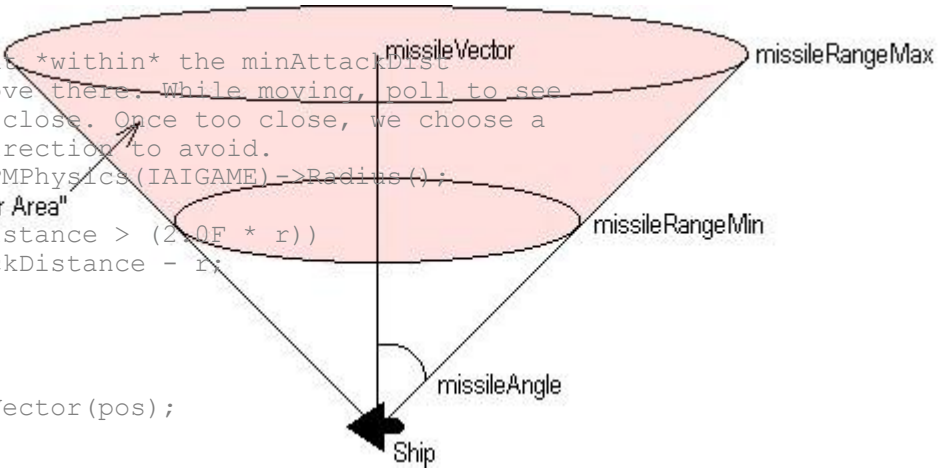


AI Data Values in *Sinistar: Unleashed*

Version 2.0002

```
state start GetClose
{
   entry code
   %{
      // Select a point *within* the minAttackDist
      // and try to move there. While moving, poll to see
      // if we're too close. Once too close, we choose a
      // "strafing" direction to avoid.
      float32 r = GetPMPhysics(IAIGAME)->Radius();
      float32 d;
      if (mMinAttackDistance > (2.0F * r))
         d = mMinAttackDistance - r;
      else
         d = r;

      cTDVector pos;
      gRandomGen.RandVector(pos);
      pos *= d;

      IAIGame* pTarget = SMGetAIGame( this, $$(MSG:SetTargetMsg_PollTarget));
      cTDVector delta = GetPMPhysics( pTarget )->WorldPosition() -
                        GetPMPhysics( IAIGAME )->WorldPosition();
      if (pos.Dot(delta) > 0.0F)
         pos *= -1.0F;

      // Move to the appropriate point
      cVectorMsg offsetMsg( IAIGAME, $$(MSG:ExcMsg_SetNoSpinOffset), pos );
      IAIGAME->ReceiveMessage( &offsetMsg, sizeof(cVectorMsg) );
      cBoolMsg worldMsg( IAIGAME, $$(MSG:ExcMsg_SetSpinOffsetWorldSpace), true
      );
      IAIGAME->ReceiveMessage( &worldMsg, sizeof(cBoolMsg) );

      receiveMessage( $$(MSG:ExcMsg_PrecisionAttackNoSpin) );
   %}

   exit code
   %{
      receiveMessage( $$(MSG:ExcMsg_Idle) );
   %}
```





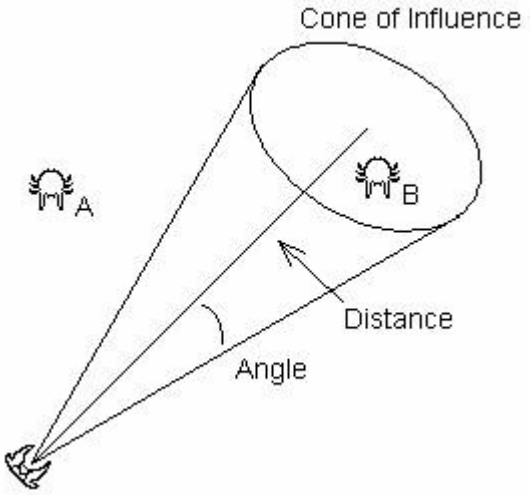


Table of Contents

1 Introduction

This document covers the AI data values in *Sinistar: Unleashed*—the parameters that allow tuning of the AI behaviors of objects and effects in the game via the .r files.

Objects have AI behaviors, which are specified in the klowns. Klowns exist for all sorts of behaviors, and a klown can call on other klowns to provide their functionality, too. For example, boss-specific behavior of bosses are controlled by the various excXX.fsm files, while general boss behaviors are controlled by the klowns of excalive.fsm, which the boss-specific klowns call on. Thus, the EXCESSION_01 boss is controlled by the DEvil_01 klown in exc01.fsm, which provides the boss's specific behaviors and calls on the klowns of excalive.fsm for general boss behaviors.

> **NOTE ON KLOWNS:**
> A klown code file, a file with the .fsm extension, can contain several klowns. There are several types of klowns:
>
> 1) The basic HFSM (Hierarchical Finite State Machine) klown is able to operate independently, can be included in other klowns, but does not call on other klowns in its operations.
>
> 2) A submachine HFSM klown cannot operate independently and must be called on by other klowns to do its thing.
>
> 3) A CHFSM (Concurrent Hierarchical Finite State Machine) klown can contain multiple HFSM klowns, including external HFSM klowns (including submachines) in its operations and having its own internal HFSM klowns. For example, the Crystal klown includes several external klowns and has an internal klown, CrystalBehavior, that contains the klown's own AI. In this document, the workings of an internal klown are documented as part of its "master" klown.
>
> 4) An internal klown is an HFSM klown that appears inside a CHFSM klown, as desribed above. It is entirely internal to its CHFSM and cannot operate as an independent or submachine HFSM. (It is often easy to set up an internal klown as an independent or submachine HFSM; check with the programmers if you want to expose an internal klown for general use.) *Warning:* Since an internal klown isn't exposed to the wide world, it does not have to be uniquely named. Thus, several internal klowns can have the same name (NormalBehavior, for example, is popular). Don't be fooled into thinking that internal klowns named the same operate the same or even similarly!
>
> See the separate document, The Klown System, for further details.)

For each klown, the klown file, specific klown, and klowns it calls upon are identified in this document as follows:

- *Klown Code File:* crystal.fsm
- *Klown:* Crystal
  —*Type:* CHFSM
  —*Uses External Klowns:* MoveEngines, OrientEngines, SetTarget, Scorable, ElectronConductive
  —*Has Internal Klown:* CrystalBehavior

Various parameters in the klown code are exposed to tuning (in .r files). These appear as AI data values in `rtAI` resources. This document covers these exposed parameters.

AI behaviors are governed through `AIDataValue` fields of `rtAI` or `rtAIOverride` resources, together with any subsidiary resources required by the `AIDataValue` fields. The general structure of `rtAI` or `rtAIOverride` resources and `AIDataValue` fields is covered in the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*, and is not repeated here.

An `rtAI` definition in a .r file must contain at least one parameter. However, in some cases, you may want to assign a klown to an AI, and the klown will work fine without any user defined parameters. In this case, include a bogus parameter to the resource definition:

- **`bogus`** (integer) is a bogus AI parameter. Although defined, the governing klown will simply ignore it.

Example:

```
defres rtAI ( BOSS1_RING )
(
   Type.Klown = ( KlownName = "Explosion" )
   AIDataValue ( "bogus", Integer, ( Val = 0 ))
)
```

Be aware of some slightly tricky items:

1. If an object doesn't have a particular klown, defining an AI parameter for that klown in the object's .r file has no effect. It does not make the klown operational for that object! (For example, defining `limitMaxRadius` does not limit the object to staying within a sphere if the object does not use the SphericalRestriction klown.) If you want to add a klown for an object, get thee to a programmery.
2. Many AI parameters have default values. A default is used when an object's AI uses a klown with such a parameter, but the parameter is not specified in the .r file. The tricky part here is that when you look at a .r file, the absence of a parameter can mean either a) the object is using the default value, or b) the object just doesn't use the parameter's klown. For example, the lack of a `limitMaxRadius` parameter for an object could mean the object does not use the SphericalRestriction klown, but it could also mean it does use the klown but with the parameter's default value.
3. If an object's klown includes another klown, remember that the included klown's AI parameters can (and often should) be defined for that object. If the parameters are not defined, their default values are used. For example, the CrystalDepot klown includes the separate Team klown, so you can use the `team` AI parameter to assign a depot to a particular team.

Many AIs involve the interaction of a ship with its target. For convenience, this document is written using the player as the target of Sporg and Distilled warriors and bosses, as this is typically the case. However, keep in mind that the AIs actually apply to any valid target. For example, a mind-controlled warrior will no longer target the player but will target its former teammates instead.

Some parameters in the klown code are hardwired into the code, and these cannot be tuned. Whenever you see in the text a fixed number that has no AI data value controlling it, that value is hardcoded and cannot be set via the .r files. For example, if the text says it takes a boss five seconds to do something, and there's no AI data value controlling that time factor, then that five seconds is hardcoded. Some (not all) of this hardwired values are noted as being such in the text.

1.1 Document Version History

The original document this document is based on is Boss, Warrior, and Other AI Behaviors in *Sinistar: Unleashed*, version 1.1012. That documented covered some 400 AI data values in the game and was generated on an as-needed basis for the game's designers and tuners.

This document is an expanded and updated version of the earlier document, with its goal to cover all AI data values in the gold version of the game (late August 1999).

1.2 Document Conventions

AI data values can be floating point, integer, string, resource, vector, or matrix values. See the AI Resources section in the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*, for details. In general:

- "Floating point" mean an AI data value uses a floating point value; "integer" means an integer value. Where appropriate, value ranges for fields are indicated, such as ">= 0, <= 100" (value must be greater than or equal to 0 and less than or equal to 100) or "-1.0, >=0.0" (value must be -1.0 or greater than or equal to 0.0 but cannot be any negative value other than -1.0). Remember that resource IDs are actually integer values and thus can be used with integer AI data values, when appropriate.
- "Resource" means the AI data value points to a separate resource; the resource type and the specific resource ID. This resource itself must be defined in a .r file (typically the same file containing the resource AI data value).
- "String," "vector," and "matrix" AI data values are also available, as covered in the cztypes.rh document.
- If an AI data value has a default value, the default is indicated.

Some time-related AI data values use integer values while others use floating point values. Be aware of the difference, as using the wrong value in your definition can have a major impact. For *Sinistar: Unleashed*, integer time units are milliseconds while floating point time units are seconds.

Other units are those used by the physics model of the game. For *Sinistar: Unleashed*, distance is in meters, velocity in meters per second, and mass in kilograms.

The following conventions are used in the syntax definitions:

- Keywords and other standard items appear in monospace text. *Example:* `AIDataValue ( "health", Integer, ( Val = 10 ))`.
- User-defined input appears in it italics like this: <#>. Enter an appropriate value of your choice in place of the entire italicized placeholder. For example, for `AIDataValue ( "health", Integer, ( Val = <#> ))`, you could enter `AIDataValue ( "health", Integer, ( Val = 10 ))`.

2 Common Object Behaviors

2.1 Basic Parameters

2.1.1 Health Klown: Health, Damage, & Armor

- *Klown Code File:* health.fsm
- *Klown:* Health
  —*Type:* Submachine
  —*Uses External Klowns:* Difficulty

2.1.1.1 Health

```
AIDataValue ( "health",   Integer,   ( Val = <#> ))
```

Ships (and other objects) have health, which is the amount of damage they can take before being killed:

- **`health`** (integer; default is 1) specifies the starting and maximum possible health of the ship. The default health is one health point. If `health` is set to -1, then the ship is invulnerable to damage.

Example:

```
AIDataValue ( "health",   Integer,   ( Val = 10 ))
```

2.1.1.2 Minimum and Maximum Health

```
AIDataValue ( "minHealth",   Float,   ( Val = <#> ))
AIDataValue ( "maxHealth",   Float,   ( Val = <#> ))
```

Ships can back have their health adjusted according to the difficulty level:

- **`minHealth`** (floating point; default is -1.0) specifies the ship's starting and maximum possible health at the lowest difficulty level.
- **`maxHealth`** (floating point; default is -1.0) specifies the ship's starting and maximum possible health at the highest difficulty level.

A ship's starting health for a specific difficulty level is scaled linearly between its `minHealth` and `maxHealth` values. If `minHealth` and `maxHealth` are not specified, the ship's health just uses the `health` parameter instead.

Example:

```
AIDataValue ( "minHealth",   Float,   ( Val =  5.0 ))
AIDataValue ( "maxHealth",   Float,   ( Val = 25.0 ))
```

2.1.1.3 Damage

```
AIDataValue ( "spangSequence",      Resource,  ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "useOffsetForDamage", Integer,   ( Val = ( <#> ))
```

The damage an object actually receives has a random factor in it and can vary randomly from 50% to 150% of the raw damage value. For example, if a shot nominally inflicts 10 points of damage, an object hit by the shot can receive any amount of damage from 5 to 15 points.

When an object takes damage, a sequence plays on it if it has a spang sequence:

- **spangSequence** (resource; no default) specifies the `rtSequence` resource that plays when an object takes damage. (This sequence is in world space; position is the spang position; orientation is set so the Y axis points outward along the collision normal.)
- **useOffsetForDamage** (integer; default is 0) specifies whether (`Val=1`) or not (`Val=0`) the spang sequence if offset from the damaged object.

Note: You can play any sequence for `spangSequence`, not just one that emits spangs. This allows you to create all sorts of effects when a ship takes damage.

Example:

```
AIDataValue ( "spangSequence",      Resource, ( Val = ( rtSequence, SEQ_TICK_DEFAULT_SPANG
                ))
AIDataValue ( "useOffsetForDamage", Integer,  ( Val = ( 0 ))
```

2.1.1.4 Ignoring Damage

```
AIDataValue ( "IgnoreSplashDamage",      Integer,   ( Val = <#> ))
AIDataValue ( "IgnoreCollisionDamage",   Integer,   ( Val = <#> ))
AIDataValue ( "IgnoreRoidDamage",        Integer,   ( Val = <#> ))
AIDataValue ( "IgnoreElectricDamage",    Integer,   ( Val = <#> ))
AIDataValue ( "IgnoreAllDamage",         Integer,   ( Val = <#> ))
```

An object can be flagged to ignore certain types of damage:

- **IgnoreSplashDamage** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`, the default) the object ignores splash damage. (Weapons that produce blast wave effects—blast waves and concussion bursts—inflict splash damage.)
- **IgnoreCollisionDamage** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`, the default) the object ignores damage from collisions in general.
- **IgnoreRoidDamage** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`, the default) the object ignores damage from collisions with asteroids. (Workers and anything else that mucks with roids should ignore roid damage.)
- **IgnoreElectricDamage** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`, the default) the object ignores damage from electricity. *Note:* The object still wicks electricity (unless `doNotWickElectricity` is set to 1, see Electricity Wicking) but just is not damaged by it.
- **IgnoreAllDamage** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`, the default) the object ignores all damage.

Example:

```
AIDataValue ( "IgnoreSplashDamage",      Integer,   ( Val = 1 ))
```

```
AIDataValue ( "IgnoreCollisionDamage",   Integer,   ( Val = 0 ))
AIDataValue ( "IgnoreRoidDamage",        Integer,   ( Val = 1 ))
AIDataValue ( "IgnoreElectricDamage",    Integer,   ( Val = 1 ))
AIDataValue ( "IgnoreAllDamage",         Integer,   ( Val = 0 ))
```

2.1.1.5 Electricity Wicking

AIDataValue ("doNotWickElectricity", Integer, (Val = *<flag>*))

Objects that are hierarchically linked to parent objects (like a turret attached to a ship) by default wicks electricity to its parent. Wicking causes some of the damage from electricity to flow to the parent object; the amount that flows is proportional to the objects' masses. For example, if a turret of mass 100 is attached to a ship of mass 1,000, then, when the turret is hit by an electric shot, 10% of the electric damage is applied to the turret and the other 90% wicks to the ship.

In some cases, it makes sense for an object not to wick electricity. For example, an orbital can be attached to a ship, but in concept it is orbiting the ship with no physical connection and thus should not wick electricity to its parent ship. For another example, turrets on really massive objects like bosses should not wick electricity, since the boss's mass will wick off almost all damage (making the turret highly resistant to electric damage)

You can make an object not wick electricity by using:

- **doNotWickElectricity** specifies whether the object does not wick electricity (`Val = 1`) or does (`Val = 0`).

Example:

```
AIDataValue ( "doNotWickElectricity", Integer, ( Val = 1 ))
```

2.1.1.6 Death and Destruction

AIDataValue ("deathSequence", Resource, (Val = (rtSequence, *<resource ID>*)))

When an object's health drops to or below zero, the object dies and a sequence plays if it has a death sequence:

- **deathSequence** (resource; no default) specifies the `rtSequence` resource that plays on an object when it dies.

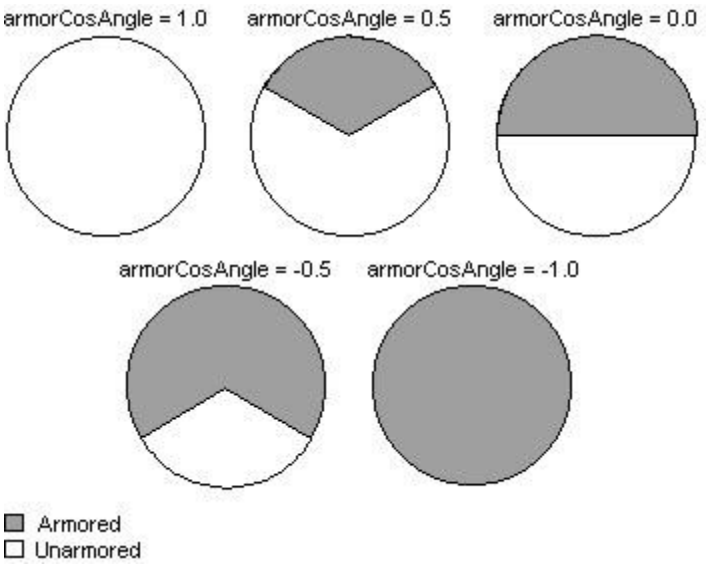
Example:

```
AIDataValue ( "deathSequence", Resource, ( Val = ( rtSequence, SEQ_TICK_DEFAULT_SPANG )))
```

2.1.1.7 Armor

```
AIDataValue ( "armorVector",    Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "armorCosAngle",  Float,    ( Val = <#> ))
AIDataValue ( "armorFactor",    Float,    ( Val = <#> ))
```

Armor can reduce or negate the damage an object takes. (It can also be used to increase damage or turn damage into health gains!) Objects can be partially or fully armored. Armor affects all damage except electric damage.



The following AI parameters control armor:

- **armorVector** (vector; default is 0.0, 0.0, 0.0) specifies whether the object has armor. When `Val = ( 0.0, 0.0, 0.0 )`, the ship does not have armor; when set to any other value (by convention, 0.0, 0.0, 1.0), the ship is armored.
- **armorCosAngle** (floating point; default is 0.0) in effect specifies how much of the object is armored. The cosine of the angle of a shot or missile striking the object is calculated and if it is greater than `armorCosAngle`, the shot or missile strikes the armored portion of the object.
- **armorFactor** (floating point; default is 1.0) is the multiplicative factor applied to the damage to calculate the actual damage. For example, at `Val = 0.5` only half the damage is applied to the object, while at `Val = 0.0` none of the damage is applied to the object. Note that `armorFactor` values above 1.0 increase the amount of damage the object takes, and negative `armorFactor` values result in negative damage, which adds health to the object.

Example:

```
AIDataValue ( "armorVector",    Vector,   ( Val = ( 0.0, 0.0, 1.0 )))
AIDataValue ( "armorCosAngle",  Float,    ( Val = -0.1 ))
AIDataValue ( "armorFactor",    Float,    ( Val =  0.1 ))
```

2.1.1.8 Obsolete: Killer Text

```
AIDataValue ( "killerFlavorText",   String,   ( Val = "<string>" ))
```

When the player ship gets killed, the Score Screen displays a text string from the AI of the object that killed the player ship:

- **killerFlavorText** (string; maximum 80 characters; default is `Killed by an unknown entity`) specifies the string that is displayed on the Score Screen.

Have a `killerFlavorText` parameter to the `rtAI` resource of each object that can kill the player. For example, for an asteroid:

```
AIDataValue ( "killerFlavorText",  String,  ( Val = "Got too friendly with an asteroid" ))
```

The `killerFlavorText` strings for enemy ships should be about death ***by collision*** with that ship. For example, "Collided with a Distilled Evil Enforcer."

The `killerFlavorText` strings for enemy shots should be about death ***by being shot down by a particular enemy***. For example, "Shot down in flames by a Sporg Guardian burst."

2.1.2 SkeletonSelectiveDestruction Klown

- *Klown Code File:* SkeletonDamage.fsm
- *Klown:* SkeletonSelectiveDestruction
  —*Type:* Submachine

```
AIDataValue ( "jointNodeContainer",   Integer,   ( Val = ( <resource ID> ))
AIDataValue ( "jointHealth",          Resource,  ( Val = ( rtJointHealth, <resource ID>
                ) ))
AIDataValue ( "jointSpangSequence",   Resource,  ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "jointDeathSequence",   Resource,  ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "damageColor",          Integer,   ( Val = ( <RGBA macro> ))
AIDataValue ( "damageTime",           Float,     ( Val = ( <#> ))
AIDataValue ( "deathFadeTime",        Float,     ( Val = ( <#> ))
```

Joints of an object can have their own health, sequences, and effects:

- **`jointNodeContainer`** (integer; default is 1) specifies the container resource ID for the joint's node.
- **`jointHealth`** (resource; no default) specifies the `rtJointHealth` resource that contains the health values for the joint (joint name, initial health, maximum health). See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtJointHealth` resources.
- **`jointSpangSequence`** (resource; no default) specifies the `rtSequence` resource that plays when the joint takes damage.
- **`jointDeathSequence`** (resource; no default) specifies the `rtSequence` resource that plays when the joint is killed.
- **`damageColor`** (RGBA macro; default is RGBA(255, 0, 0, 255)) specifies the RGBA color (and can include alpha) that flashes on the joint when the joint takes damage.
- **`damageTime`** (floating point; default is 0.1) specifies the time (in seconds) the `damageColor` plays.
- **`deathFadeTime`** (floating point; default is 1.0) specifies the time (in seconds) over which the joint fades out and disappears when it is killed.

Example:

```
AIDataValue ( "jointNodeContainer",   Integer,   ( Val = ( EXAMPLE_JOINT ))
AIDataValue ( "jointHealth",          Resource,  ( Val = ( rtJointHealth, EXAMPLE_JOINT
                ) ))
AIDataValue ( "jointSpangSequence",   Resource,  ( Val = ( rtSequence, SEQ_JOINT_SPANG )))
AIDataValue ( "jointDeathSequence",   Resource,  ( Val = ( rtSequence, SEQ_JOINT_DEATH )))
AIDataValue ( "damageColor",          Integer,   ( Val = ( RGBA( 100, 100, 100, 255 ) ))
AIDataValue ( "damageTime",           Float,     ( Val = ( 0.1 ))
AIDataValue ( "deathFadeTime",        Float,     ( Val = ( 1.0 ))
```

2.1.3 Teams and Mind Control

2.1.3.1 Team Klown

- *Klown Code File:* team.fsm
- *Klown:* Team
  —*Type:* Submachine

```
AIDataValue ( "team",              Integer,    ( Val = <resource ID> ))
AIDataValue ( "cantChangeTeams",   Integer,    ( Val = <#> ))
AIDataValue ( "teamColorOverride", Integer,    ( Val = <RGB macro> ))
```

Ships (and other objects) are organized by team. A ship on a specific team will not fire at anything else on its team, even if its sensors can detect the object as a valid target. Set team parameters using:

- **team** (integer; default is 0) specifies the resource ID of the ship's team. There are three core teams: `TEAM_TIMMY` (or 0) for the player's team (but see notes), `TEAM_TICK` (or 1) for the Sporg team, and `TEAM_EVIL` (or 2) for the Distilled Evil team. (Bosses will be in either `TEAM_TICK` or `TEAM_EVIL`, depending upon whether they are on a Sporg or DE level.) On occasion, other teams have been defined (liked a cloaked team for all ships using cloaks), but none remain in use at present.
- **cantChangeTeams** (integer; 0 or 1; default is 0) specifies whether or not the object can change teams (such as when it is hit by a mind control shot). If `Val = 1`, then the object cannot change teams; if `Val = 0` (the default), then it can.
- **teamColorOverride** (integer; RGB color macro; no default) specifies the color to in place of the object's team color, for various game features such as the long-range and short-range radars. (In *Sinistar*, this was used to tint workers different from warriors.)

Notes:

1. Technically, when `Val = 0` for `team`, the ship or object is on no team at all and can fire on anything its sensors detect as a valid target.
2. The Team klown also handles some mechanics of cloaking; see Cloaks.

Example:

```
AIDataValue ( "team",              Integer,    ( Val = TEAM_TICK ))
AIDataValue ( "cantChangeTeams",   Integer,    ( Val = 1 ))
```

2.1.3.2 MindControl Klown

- *Klown Code File:* mindcontrol.fsm
- *Klown:* MindControl
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Fader, Health
  —*Has Internal Klown:* MindControlGuts

```
AIDataValue ( "team",                  Integer,      ( Val = <#> ))
AIDataValue ( "mindControlSequence",   Resource,   ( Val = ( rtSequence, <resource ID> )))
```

The MindControl klown allows a shot, when it strikes certain targets, to change the team of the target:

- **team** (integer; no default) is the team the target changes to.

- **mindControlSequence** (resource; no default) is the `rtSequence` resource that plays on a mind-controlled target. (In *Sinistar*, the model of a mind-controlled object does not change, so this sequence allows a visual effect to be played on the object, to distinguish it from similar objects that are no mind controlled.)

Example:

```
AIDataValue ( "team",             Integer,    ( Val = TEAM_TIMMY ))
AIDataValue ( "mindControlSeq",   Resource,   ( Val = ( rtSequence, PS_MINDCONTROL_SEQ2
                ) ))
```

2.1.4 Scoring

2.1.4.1 Score Klown

- *Klown Code File:* score.fsm
- *Klown:* Score
  —*Type:* Submachine

Anything that can accumulate a score (in *Sinistar*, the player) must have this klown. It no longer has any AI parameters.

2.1.4.1.a Obsolete: Score Value

Note: This AI data value is now obsolete and is no longer used.

```
AIDataValue ( "scoreValue",   Float,   ( Val = <#> ))
```

Objects can be worth points in the scoring system:

- **scoreValue** specifies the points the object is worth in the game's scoring system. The player scores these points when he does the appropriate action to the object (e.g., destroying an enemy warrior gets its score value).

Example:

```
AIDataValue ( "scoreValue",   Float,   ( Val = 10000.0 ))
```

2.1.4.2 ScoreIndicator Klown

- *Klown Code File:* score.fsm
- *Klown:* ScoreIndicator
  —*Type:* Submachine

```
AIDataValue ( "scoreInfo",        Resource, ( Val = ( rtScoreInfo, <resource ID> )))
AIDataValue ( "missionBonusInfo", Resource, ( Val = ( rtMissionBonusInfo, <resource ID>
                ) ))
```

`rtScoreInfo` resources assign specific scores for each of the game's score categories. For a level to have access to these scores, place the following parameter in the level's AI (`rtAI ( MISSION_SETUP )` in the level's .r file):

- **scoreInfo** (resource; no default) specifies the `rtScoreInfo` resource that contains the specific scores for each score category.

`rtMissionBonusInfo` resources contain information on any bonus for levels. For a level to have access to this data, place the following parameter in the level's AI (`rtAI ( MISSION_SETUP )` in the level's .r file):

- **`missionBonusInfo`** (resource; no default) specifies the `rtMissionBonusInfo` resource that contains the information about the failure of the mission on the level, the success of the mission, and the destruction of the gate. This formation can contain messages to be displayed on the Score Screen, scoring points, and other data. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on what `rtScoreInfo` resources contain and how to define them.

Example:

```
AIDataValue ( "scoreInfo",        Resource, ( Val = ( rtScoreInfo, MISSION_SCORES )))
AIDataValue ( "missionBonusInfo", Resource, ( Val = ( rtMissionBonusInfo, MISSION_SCORES
                ))
```

Note: In theory, each level's `scoreInfo` parameter can point to its own `rtScoreInfo` resource. (That way, you can have each level score the categories different from one another.) However, the game is currently set up so that each level uses the same `rtScoreInfo` resource (`rtScoreInfo ( MISSION_SCORES )` in ui.r), and it is expected that the final game will indeed work this way.)

In ui.r, the `rtScoreInfo` resource lists the name of each score category, how many points to award for each kill of an enemy in a particular score category, and a resource ID of an icon to be used to display the kill in the final score screen. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtScoreInfo` resources.

Example:

```
// Items in this list must appear in the order of their categories
// listed in missionid.rh...
defres rtScoreInfo ( MISSION_SCORES )
(
   ScoreCategory[] = ( Name = "Slave Drudge"     Icon = 0 Score = 50 )
   ScoreCategory[] = ( Name = "Light Turret"     Icon = 0 Score = 100 )
   ScoreCategory[] = ( Name = "Strike Warrior"   Icon = 0 Score = 100 )
   ScoreCategory[] = ( Name = "Squadron Warrior" Icon = 0 Score = 150 )
   ScoreCategory[] = ( Name = "Heavy Fighter"    Icon = 0 Score = 200 )
   ScoreCategory[] = ( Name = "Attack Carrier"   Icon = 0 Score = 500 )
)
```

2.1.4.3 Scorable Klown

- *Klown Code File:* score.fsm
- *Klown:* Scorable
  —*Type:* Submachine

```
AIDataValue ( "scoreCategory",   Integer,   ( Val = <resource ID> ))
```

In the score system, objects have a score category (instead of a score value per the old system, see Score Klown). The score category organizes objects into categories for scoring purposes. To set the score category for an object, add the following parameter to its `rtAI` resource:

- **`scoreCategory`** (default is no category) specifies the category of the object for scoring purposes.

Note: The score category is a game-wide method to groups objects into categories without assigning specific scores to them. See Score and Mission Bonus Info for assigning specific scores.

Example:

```
AIDataValue ( "scoreCategory",   Integer,   ( Val = SCORECAT_TURRET ))
```

The list of valid score categories is located in missionID.rh. At one point, it was:

Category	*Integer Value*
SCORECAT_WORKER	0
SCORECAT_TURRET	1
SCORECAT_WARRIOR4	2
SCORECAT_WARRIOR1	3
SCORECAT_WARRIOR2	4
SCORECAT_WARRIOR3	5
SCORECAT_APC	6
SCORECAT_DE_TURRET	7
SCORECAT_DE_WARRIOR1	8
SCORECAT_DE_WARRIOR2	9
SCORECAT_DE_WARRIOR3	10
SCORECAT_DE_WARRIOR4	11
SCORECAT_DE_APC	12

2.1.4.4 ScoreVector Klown

- *Klown Code File:* score.fsm
- *Klown:* ScoreVector
  —*Type:* Submachine

This submachine is attached to shots and missiles, so that kills by these can be attributed to the right scorer. It has no AI parameters.

2.1.5 ShipModel Klown

- *Klown Code File:* shipmodel.fsm
- *Klown:* ShipModel
  —*Type:* Submachine

```
AIDataValue ( "thirdpersonModel",   Integer,   ( Val = <resource ID> ))
AIDataValue ( "firstpersonModel",   Integer,   ( Val = <resource ID> ))
AIDataValue ( "cloakModel",         Integer,   ( Val = <resource ID> ))
AIDataValue ( "modelFadeTime",      Float,     ( Val = <#> ))
AIDataValue ( "cloakFadeTime",      Float,     ( Val = <#> ))
AIDataValue ( "decloakFadeTime",    Float,     ( Val = <#> ))
```

These parameters control aspects of the ship's model:

- **thirdPersonModel** (resource ID; no default) specifies the resource ID of the third-person model of the ship.
- **firstPersonModel** (resource ID; no default) specifies the resource ID of the first-person model of the ship. (This is used for the player ship, when the player switches from third person view to first person view.)
- **cloakModel** (resource ID; no default) specifies the resource ID of the cloaked model of the ship (if the ship has a cloak).

- **modelFadeTime** (floating point; default is 1.0) specifies the time (in seconds) it takes to fade between first- and third-person models.
- **cloakFadeTime** (floating point; default is `modelFadeTime`) specifies the time (in seconds) it takes to fade between cloaked and non-cloaked models when cloaking.
- **decloakFadeTime** (floating point; default is `modelFadeTime`) specifies the time (in seconds) it takes to fade between cloaked and non-cloaked models when decloaking.

Example:

```
AIDataValue ( "firstPersonModel",   Integer,   ( Val = FIRSTPERSON_T1 ))
AIDataValue ( "thirdPersonModel",   Integer,   ( Val = TIMMY1         ))
AIDataValue ( "cloakModel",         Integer,   ( Val = CLOAK_T1       ))
AIDataValue ( "modelFadeTime",      Float,     ( Val = 1.0 ))
AIDataValue ( "cloakFadeTime",      Float,     ( Val = 4.0 ))
AIDataValue ( "decloakFadeTime",    Float,     ( Val = 2.0 ))
```

2.1.6 AutoHealth Klown

- *Klown Code File:* health.fsm
- *Klown:* AutoHealth
  —*Type:* Submachine
  —*Uses External Klowns:* PollFast

2.1.6.1 Current System

```
AIDataValue ( "easyHealthPerSec",    Float,    ( Val = <#> ))
AIDataValue ( "hardHealthPerSec",    Float,    ( Val = <#> ))
AIDataValue ( "easyCrystalPerSec",   Float,    ( Val = <#> ))
AIDataValue ( "hardCrystalPerSec",   Float,    ( Val = <#> ))
```

A ship that uses crystals can use the AutoHealth klown to automatically spend crystals to heal itself. The klown checks the ship's health continuously and spends crystals to regenerate health as long as the ship's current health is less than its maximum health. (The amount of health gained and the crystal cost depend upon the difficulty level, as explained below.) The following parameters control autohealth:

- **easyHealthPerSec** (floating point; default is 0.0) and **hardHealthPerSec** (floating point; default is 0.0) are the amount of health the player ship regenerates per second at the minimum ("easy," difficulty level 0.0) and maximum ("hard," difficulty level 1.0) difficulty levels. (Note that "easy" and "hard" in these parameters do ***not*** correspond to the difficulty levels the player chooses when selecting the difficulty level.) The actual health per second value used for a player-set difficulty level is scaled between the `easyHealthPerSec` and `hardHealthPerSec` values. The player ship regenerates health at this rate as long as it is below its maximum health and has crystals to spend.
- **easyCrystalPerSec** (floating point; default is 0.0) and **hardCrystalPerSec** (floating point; default is 0.0) are the amount of crystals the player ship spends per second when regenerating health at the minimum ("easy," difficulty level 0.0) and maximum ("hard," difficulty level 1.0) difficulty levels. (Note that "easy" and "hard" in these parameters do ***not*** correspond to the difficulty levels the player chooses when selecting the difficulty level.) The actual crystals per second value used for a player-set difficulty level is scaled between the `easyCrystalPerSec` and `hardCrystalPerSec` values.

Note that the above values can be floating point value because these are just rates over time. Health and crystals are still changed in integer amounts.

Example:

```
AIDataValue ( "easyHealthPerSec",    Float,    ( Val = 12.0 ))
AIDataValue ( "hardHealthPerSec",    Float,    ( Val = 12.0 ))
AIDataValue ( "easyCrystalPerSec",   Float,    ( Val =  1.0 ))
AIDataValue ( "hardCrystalPerSec",   Float,    ( Val =  1.0 ))
```

2.1.6.2 Obsolete System

```
AIDataValue ( "minAutoHealth",    Integer,   ( Val = <#> ))
AIDataValue ( "maxAutoHealth",    Integer,   ( Val = <#> ))
AIDataValue ( "autoHealthTime",   Float,     ( Val = <#> ))
```

The player ship automatically spends crystals to heal itself. The klown checks the ship's health once every interval of time and expands a crystal if the ship's current health is less than its maximum health. (The amount of health added per crystal spent depends upon the difficulty level, as explained below.) The following parameters control autohealth:

- **minAutoHealth** (floating point; default is 1) specifies the amount of health (per crystal spent) the ship receives at the lowest difficulty level.
- **maxAutoHealth** (floating point; default is 1) specifies the amount of health (per crystal spent) the ship receives at the highest difficulty level.
- **autoHealthTime** (floating point; default is 1.0) specifies the time interval (in seconds) at which the autohealth klown checks the ship's health status. If the ship's health is not at maximum, one crystal is expended to restore some health. *Note:* While `autoHealthTime` is a floating point variable, as of this writing the AutoHealth klown in effect rounds it off to the nearest integer and then uses the integer value.

For a specific difficulty level, a ship's health gain per crystal is scaled linearly between its `minAutoHealth` and `maxAutoHealth` values.

Notes: 1) If the ship has no crystals when the autohealth klown does it thing, the ship does not get any health. 2) The klown will spend a crystal for health, even if most of the health will be wasted because the ship is near by not at full health. To compensate for this, the default time interval at which the klown does its check is relative large. It is being considered to change the klown so that it will only spend a crystal if it can get full value for the crystal, thereby allowing the time interval to be dropped. Alternatively, "wasting crystals" at low time intervals may turn out to be acceptable.

Example:

```
AIDataValue ( "minAutoHealth",    Integer,   ( Val = 5 ))
AIDataValue ( "maxAutoHealth",    Integer,   ( Val = 1 ))
AIDataValue ( "autoHealthTime",   Float,     ( Val = 1.0 ))
```

2.2 Sensing

2.2.1 Sensors

- *Klown Code Files:* various files
- *Klowns:* various klowns

```
AIDataValue ( "antiMissileSensor",   Resource,   ( Val = ( rtBodySensor, <resource ID> )))
```

Ships can have a variety of sensors, which can detect various objects and accordingly trigger various behaviors. A sensor parameter specifies an `rtBodySensor` resource named *<resource ID>*, which defines the abilities of the sensor such as its range and what objects it can detect. (See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtBodySensor` resources.) The general form of a sensors is:

- ***<name>*Sensor** (resource; default is no sensor) specifies the sensor that an object uses for a particular purpose, where <name> indicates in some fashion the purpose of the sensor. For example, **enemySensor** is a sensor ships use to detect enemies.

The specific sensors in the game are covered with the klowns that call on the sensor.

Example:

```
AIDataValue ( "anitMissileSensor",   Resource, ( Val = ( rtBodySensor, EX00_ANTIMISSILE
                  )))

defres rtBodySensor ( EXC01_ANTIMISSILE )
(
   Type.Simple =
   (
      MaxDistance = 1000.0
      SenseType   = kSenseMissile
   )
)
```

2.2.2 Sensing Types

Objects are categorized in various sensing types for uses with sensors (and other game mechanics):

- **kSenseWorker** for workers.
- **kSenseWarrior** for warriors.
- **kSenseRoid** for asteroids.
- **kSenseCrystal** for crystals.
- **kSenseTurret** senses turrets.
- **kSenseGarbage** senses space garbage like the hulks of destroyed ships.
- **kSensePlayer** for the player ship.
- **kSenseOffering** senses offerings.
- **kSenseMonster** senses the Sinistar.
- **kSenseEgg** senses boss eggs and jumpgates.
- **kSenseShot** senses unguided shots (anything that uses the shot klown).
- **kSenseMissile** senses guided missiles, Sinibombs, and anything else that uses the missile klown.
- **kSenseAll** senses all of the above items.

Note: Another type of object, the "boring type" (`kBoringType`), exists as a type exclusive of all other types (including `kSenseAll`). Boring types are used for cloaked ships, for orbitals so that they won't show up on sensors separate from their parent objects, for dummy turrets, etc.

2.3 Movement & Orientation

2.3.1 Engines

There are a wide variety of engine klowns, to allow for the various types of ship movement in the game: interial lateral and spin movement, inertialess lateral and spin movement, inertial circling, inertialess circling, spin movement without lateral movement, etc. However, the vast majority of engine klowns are just variations of two klowns, MatchEngines and MatchEnginesInstantly.

2.3.1.1 MatchEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* MatchEngines
  —*Type:* Submachine

```
AIDataValue ( "normalMotion",   Resource,  ( Val = ( rtBodyMotion,  <resource ID> )))
AIDataValue ( "normalControl",  Resource,  ( Val = ( rtBodyControl, <resource ID> )))
```

The MatchEngines klown allows a ship to use inertial lateral movement and spin. Two parameters are used:

- **normalMotion** (resource; no default) specifies the `rtBodyMotion` resource that contains the ship's basic movement abilities.
- **normalControl** (resource; no default) specifies the `rtBodyControl` resource that refines the ship's basic movement abilities specified in its `rtBodyMotion` resource.

Example:

```
AIDataValue ( "normalMotion",    Resource,   ( Val = ( rtBodyMotion,  EXCESSION_01 )))
AIDataValue ( "normalControl",   Resource,   ( Val = ( rtBodyControl, EXCESSION_01 )))

defres rtBodyMotion( EXCESSION_01 )
(
   MaxVelocity =    (   200.0,  300.0,  200.0 )
   MinVelocity =    (  -200.0, -300.0, -200.0 )
   MaxAngVelocity = (     1.2,    1.2,   1.2 )
   MinAngVelocity = (    -1.2,   -1.2,  -1.2 )
)

defres rtBodyControl ( EXCESSION_01 )
(
   LatGainAccel = (  0.7,   0.7,   0.7  )
   LatGainDecel = (  0.5,   0.5,   0.5  )
   LatBiasAccel = (  0.0,   0.0,   0.0  )
   LatStdDev    = (  0.2,   0.2,   0.2  )

   AngGainAccel = (  0.7,   0.7,   0.7  )
   AngGainDecel = (  1.0,   1.0,   1.0  )
   AngBiasAccel = (  0.0,   0.0,   0.0  )
   AngStdDev    = (  0.2,   0.2,   0.2  )
)
```

2.3.1.2 MatchEnginesInstantly Klown

- *Klown Code File:* ship.fsm
- *Klown:* MatchEnginesInstantly
  —*Type:* Submachine

```
AIDataValue ( "normalMotion",   Resource,  ( Val = ( rtBodyMotion,  <resource ID> )))
```

The MatchEngines klown allows a ship to use inertialess (UFO-like) lateral movement and spin. One parameters is used:

- **`normalMotion`** (resource; no default) specifies the `rtBodyMotion` resource that contains the ship's basic movement abilities.

Example:

```
AIDataValue ( "normalMotion",    Resource,   ( Val = ( rtBodyMotion,  EXCESSION_24 )))
```

2.3.1.3 PlayerInputEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* PlayerInputEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

The PlayerInputEngines klown allows a player to control a ship, using input from the keyboard, mouse, or joystick. The klown also uses turn sensitivity (which is set in the user interface, not by an AI parameter) and force feedback (for force feedback joysticks).

2.3.1.4 MoveNoSpinEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* MoveNoSpinEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

The MoveNoSpinEngines klown allows lateral movement without spin. The klown has no AI parameters of its own.

2.3.1.5 CircleEnginesNoSpin Klown

- *Klown Code File:* ship.fsm
- *Klown:* CircleEnginesNoSpin
  —*Type:* Submachine

```
AIDataValue ( "circleDistance",      Float,    ( Val = <#> ))
```

The CircleEnginesNoSpin klown allows circling movement without spin. Circling requires only a single parameter:

- **`circleDistance`** (floating point; default is 0.0 in each klown) defines the distance in meters from a position around which the ship tries to circle.

Example:

```
AIDataValue ( "circleDistance",      Float,    ( Val = 800.0 ))
```

2.3.1.6 MoveEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* MoveEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = <#> ))
```

The MoveEngines klown allows a ship to move toward a target (like another ship). The klown uses a parameter that allows it to lead its target, so that it can move toward it more accurately:

- **projectileSpeed** (floating point; default is 0.0) is the distance (in meters) the klown leads its target.

Example:

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = 600.0 ))
```

2.3.1.7 MoveEnginesWithLat Klown

- *Klown Code File:* ship.fsm
- *Klown:* MoveEnginesWithLat
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = <#> ))
```

The MoveEnginesWithLat klown allows a ship to move toward a target (like another ship), using a preferential lateral direction. The klown uses a parameter that allows it to lead its target, so that it can move toward it more accurately:

- **projectileSpeed** (floating point; default is 0.0) is the distance (in meters) the klown leads its target.

Example:

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = 600.0 ))
```

2.3.1.8 OrientEnginesInternal Klown

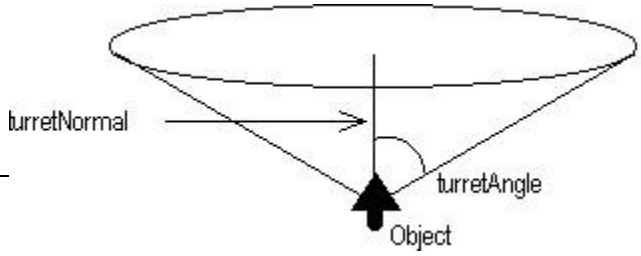
- *Klown Code File:* ship.fsm
- *Klown:* OrientEnginesInternal
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "projectileSpeed", Float,    ( Val = <#> ))
AIDataValue ( "turretNormal",    Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "turretAngle",     Float,    ( Val = <#> ))
AIDataValue ( "orientOffset",    Vector,   ( Val = ( <#>, <#>, <#> )))
```

The OrientEnginesInternal klown allows a ship to orient (spin) but not to move laterally—like a turret. The klown uses a parameter that allows it to lead its target, so that it can orient toward it more accurately:

- **projectileSpeed** (floating point; default is 0.0) is the distance (in meters) the klown leads its target (so that it can orient toward it more accurately).

The following parameters control the basic orientation of a turret and any object that can orient like a turret. (Turret-like



orientation is used to point an object at a target for subsequent movement or firing purposes. Note that it is in essence an aiming mechanism.) The parameters define a cone through which the object orients:

- **turretNormal** (vector; default is 0.0, 0.0, 0.0) specifies the normal direction in which the object faces for turret-like orientation purposes.
- **turretAngle** (floating point; default is 180.0) defines the cone angle (in degrees) from the normal within which the object can orient.
- **orientOffset** (vector; no default) specifies an orientation/aiming offset for the turret. This allows the object to be oriented at an offset from its normal direction, which is useful to correct for things like turrets with weapons that are parallel to but offset slightly from the object's Y-axis.

Example:

```
AIDataValue ( "projectileSpeed", Float,    ( Val = 600.0 ))
AIDataValue ( "turretNormal",    Vector,   ( Val = ( 0.0,0.0, 0.0 )))
AIDataValue ( "turretAngle",     Float,    ( Val = 180.0 ))
AIDataValue ( "orientOffset",    Vector,   ( Val = ( 0.0, 0.0, 5.0 )))
```

2.3.1.9 OrientEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* OrientEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines, OrientEnginesInternal

The OrientEngines klown uses the OrientEnginesInternal klown to allow it to spin like a turret, and the MatchEngines klown, so that it can also move laterally through space.

2.3.1.10 AccurateOrientEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* AccurateOrientEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEnginesInstantly, OrientEnginesInternal

The AccurateOrientEngines klown uses the OrientEnginesInternal klown to allow it to spin like a turret, and the MatchEnginesInstantly klown, so that it can also move laterally through space like a UFO.

2.3.1.11 MissileEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* MissileEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = <#> ))
```

The MissileEngines klown allows long-range missiles to move. The klown uses a parameter that allows it to lead its target, so that it can move toward it more accurately:

- **projectileSpeed** (floating point; default is 0.0) is the distance (in meters) the klown leads its target.

Example:

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = 600.0 ))
```

2.3.1.12 SRMissileEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* SRMissileEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = <#> ))
```

The SRMissileEngines klown allows long-range missiles to move. The klown uses a parameter that allows it to lead its target, so that it can move toward it more accurately:

- **`projectileSpeed`** (floating point; default is 0.0) is the distance (in meters) the klown leads its target.

Example:

```
AIDataValue ( "projectileSpeed",    Float,    ( Val = 600.0 ))
```

2.3.1.13 CircleEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* CircleEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "circleDistance",      Float,    ( Val = <#> ))
```

The CircleEngines klown allows inertial circling movement. Circling requires only a single parameter:

- **`circleDistance`** (floating point; default is 0.0 in each klown) defines the distance in meters from a position around which the ship tries to circle.

Example:

```
AIDataValue ( "circleDistance",      Float,    ( Val = 800.0 ))
```

2.3.1.14 BarrelRollEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* BarrelRollEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "barrelSpinEffort",   Float, ( Val = <#> ))
AIDataValue ( "barrelStrafeEffort", Float, ( Val = <#> ))
```

The BarrelRollEngines klown enable a ship to make a barrel roll, which is controlled by the following parameters:

- **`barrelSpinEffort`** (floating point; >= -1.0, <= 1.0; default is 0.2) specifies the fraction of the ship's engine power used to perform the roll. Negative numbers cause the ship to roll counterclockwise; positive numbers clockwise. (A setting of 0.0 causes no roll.)

- **barrelStrafeEffort** (floating point; >= -1.0, <= 1.0; default is 0.0) specifies the fraction of the ship's engine power used to use strafing movement to move in a circle.

Example:

```
AIDataValue ( "barrelSpinEffort",   Float, ( Val = 0.2 ))
AIDataValue ( "barrelStrafeEffort", Float, ( Val = 0.0 ))
```

2.3.1.15 RollTowardEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* RollTowardEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

```
AIDataValue ( "rollEffort",          Float,    ( Val = <#> ))
```

This engine klown is used by ships that have "battleship turret" broadsides (turrets along their sides). It enables the ship to roll and bring each of its broadsides to bear:

- **rollEffort** (floating point; >= 0.0, <= 1.0; default is 0.2) specifies the amount of effort the boss uses as it rolls and tries to aim its broadside at the player. The effort is the fraction of the ship's engine power used to perform the roll.

Example:

```
AIDataValue ( "rollEffort",          Float,    ( Val =    0.2 ))
```

2.3.1.16 SpinEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* SpinEngines
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines

The SpinEngines klown is used for constant spin behaviors.

2.3.1.17 OrbitalEngines Klown

- *Klown Code File:* ship.fsm
- *Klown:* SpinEngines
  —*Type:* Submachine

```
AIDataValue ( "orbitRadius",   Float,    ( Val = <#> ))
AIDataValue ( "orbitIdle",     Float,    ( Val = <#> ))
AIDataValue ( "orbitRateA",    Float,    ( Val = <#> ))
AIDataValue ( "orientAxisA",   Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "orbitRateB",    Float,    ( Val = <#> ))
AIDataValue ( "orientAxisB",   Vector,   ( Val = ( <#>, <#>, <#> )))
```

Note: Some of the parameters do things that are not readily apparent in the code, particularly how `orbitIdle` works and how an orbital switches between its two possible orbits. If you need to know about these items, let me know and I'll check around further on them.

SpinEngines allow objects (like orbitals) can orbit their parent objects and can have two different orbital orientations:

- **orbitRadius** (floating point; no default) specifies the distance (in meters) from the center of the parent object at which the object orbits.
- **orbitIdle** (floating point; no default) specifies something that is not immediately evident from the code.
- **orbitRateA** (floating point; default is 0.0) specifies the rate (in meters per second) at which the object orbits in its first orbital orientation.
- **orientAxisA** (vector; default is 0.0, 1.0, 0.0) specifies the object's first orbital orientation.
- **orbitRateB** (floating point; default is 0.0) specifies the rate (in meters per second) at which the object orbits in its second orbital orientation.
- **orientAxisB** (vector; default is 0.0, 1.0, 0.0) specifies the object's second orbital orientation.

Example:

```
AIDataValue ( "orbitRadius",   Float,    ( Val = 300.0 ))
AIDataValue ( "orbitIdle",     Float,    ( Val =   0.7 ))
AIDataValue ( "orbitRateA",    Float,    ( Val =   3.0 ))
AIDataValue ( "orbitAxisA",    Vector,   ( Val = ( 2.0, 0.0, 1.0 )))
AIDataValue ( "orbitRateB",    Float,    ( Val =   0.025 ))
AIDataValue ( "orbitAxisB",    Vector,   ( Val = ( 1.0, 0.0, 0.0 )))
```

2.3.2 Spherical Movement Restrictions

- *Klown Code File:* restriction.fsm
- *Klown:* SphericalRestriction
  —*Type:* Submachine

```
AIDataValue ( "limitCenter",      Vector,   ( Val = ( <#> <#>, <#> )))
AIDataValue ( "limitMinRadius",   Float,    ( Val = <#> ))
AIDataValue ( "limitMaxRadius",   Float,    ( Val = <#> ))
```

Ships can be restricted in their movement to all or part of a sphere, as follows:

- **limitCenter** (vector; default is 0.0, 0.0, 0.0) specifies in world coordinates the center of the sphere.
- **limitMinRadius** (floating point; default is 0.0) specifies the minimum radius of the sphere, within which the ship will not enter.
- **limitMaxRadius** (floating point; default is 0.0) specifies the maximum radius of the sphere, beyond which the ship will not enter.

Example:

```
AIDataValue ( "limitCenter",      Vector,   ( Val = ( 0.0, 0.0, 0.0 )))
AIDataValue ( "limitMinRadius",   Float,    ( Val = 4500.0 ))
AIDataValue ( "limitMaxRadius",   Float,    ( Val = 5000.0 ))
```

2.3.3 Avoidance Klown: Avoidance & Mass Ratios

- *Klown Code File:* avoid.fsm
- *Klown:* Avoidance
  —*Type:* Submachine
  —*Uses External Klown:* MatchEngines, PollUpdate

```
AIDataValue ( "avoidSensor",          Resource, ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "avoidMassRatio",       Float,    ( Val = <#> ))
```

Ships try to avoid colliding with objects their avoidance sensors detect. In general, ships with avoidance ignore worrying about any object with a mass five times less than that of the ship. However, it is possible to set this mass ratio:

- **avoidSensor** (resource; default is no sensor) specifies the type of detector a ship can use to detect objects (like asteroids) they wish to avoid colliding with. Avoidance sensors are used by most types of ships. See Sensors for more details on sensors.
- **avoidMassRatio** (floating point; default is 5.0) specifies mass ratio (ship's mass/collideable object's mass) below which the ship doesn't bother trying to avoid a collision with the object.

Note: `avoidMassRatio` is mostly used for bosses so that their avoidance doesn't kick in when trying to ram the player.

Example:

```
AIDataValue ( "avoidSensor",        Resource, ( Val = ( rtBodySensor, EX01_ANTIMISSILE ))
AIDataValue ( "avoidMassRatio",     Float,    ( Val = 2.0 ))
```

2.4 Attachments

- *Klown Code File:* attached.fsm
- *Klown:* Attached
  —*Type:* Submachine

The Attached klown allows objects with ports to have subobjects attached to these ports at runtime. This allows objects like turrets (including orbital turrets) and missiles to be defined separately from the parent object and attached as needed.

```
AIDataValue ( "turretList",   Resource,   ( Val = ( rtTurretInfo, <resource ID> )))
AIDataValue ( "missileList",  Resource,   ( Val = ( rtTurretInfo, <resource ID> )))
```

An object can have a parameters that points to a separate `rtTurretInfo` resource for turrets or missiles that are attached to the object:

- **turretList** specifies the `rtTurretInfo` resource that contains the list of turrets attached to the object.
- **missileList** (resource; no default) specifies an `rtTurretInfo` resource. This resource contains the list of missiles and/or mines that are attached to the object. (Note that these are missiles attached directly to the object. Guns, including guns in attached turrets, can also fire missiles.)

See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtTurretInfo` resources.

Example:

```
AIDataValue ( "turretList",   Resource,   ( Val = ( rtTurretInfo, EXCESSION_03 )))
AIDataValue ( "missileList",  Resource,   ( Val = ( rtTurretInfo, FIGHTER )))

defres rtTurretInfo ( EXCESSION_03 )
(
   TurretPorts[] =
   (
      Name   = "photon01"
```

```
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon02"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon03"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon04"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon05"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon06"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon07"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
   TurretPorts[] =
   (
      Name   = "photon08"
      ResID  = EXC03_TURRET_PHOTON
      Angle  = 15
      Normal = ( 0.0, 1.0, 0.0 )
   )
)
```

Note: **Multiple turrets on the same port are allowed.** When specifying attached turrets, you can have models with multiple turret attributes with the same name (for example, you can have a 3D Studio Max model with multiple boxes labeled "`attribute=turret-gunarm`"). When the turrets are defined in the `rtTurretInfo` data as shown below, a turret is attached to ***all*** turret ports that have the specified name. Also, the turret normals (the central direction around which the turrets' movement is restricted) can be specified in the local coordinate system of the turret by adding `LocalNormal = 1` to the turret port info, also shown below.

```
defres rtTurretInfo ( EXCESSION_05 )
(
   TurretPorts[] =
   (
      Name        = "gunarm"
      ResID       = EXC05_TURRET_FRONT
      Normal      = ( 0.0, 1.0, 0 )
      LocalNormal = 1
      Angle       = 5
      Tag         = 0
   )
)
```

2.5 Turrets

Turrets can be subobjects attached to parent objects (via the Attached klown) or can be independent shiplike objects. Attached turrets can include ones that appear physically attached to their parent and ones that orbit their parent.

Dummy turrets (turrets with no radius) can be created and attached to objects. Dummy turrets are invisible (both visually since they have no radius and to sensors since they are automatically classified as boring type, `kBoringType`) but have the full abilities of turrets. Dummy turrets are handy for things like assigning weapons to items (like spikes on a boss) when you don't want visible turrets attached to the items.

The overall actions of turrets are controlled by the various turret klowns (TurretCombat for core turret functionality plus other turret klowns).

2.5.1 TurretCombat Klown

- *Klown Code File:* turret.fsm
- *Klown:* TurretCombat
  —*Type:* Submachine HFSM

The TurretCombat klown is a submachine that controls the common operations of most turrets, including invisible (aka dummy) turrets. A turret that has a radius of 0.0 is an invisible turret and is assigned `kBoringType` (so that it won't show up on sensors or radars), while other turrets are assigned `kTurretType` (but see 2.4.1.2 for orbital turrets). Other than type, an invisible turret works just like other turrets—it's an object that can have weapons and other attributes.

An active turret searches for targets and orients toward them. It periodically "parks" if it cannot find a target or if it is in a collision, reactivating later. A turret tracking a target periodically checks to see if the target is outside its tracking range or if there is a better targer around than the one it is tracking.

Note that the TurretCombat klown itself doesn't fire weapons or use fire control or so on. These functions are handled by other klowns. For example, the Turret klown uses TurretCombat along with other klowns such as Weapons and FireControl.

2.5.1.1 Basic Parameters

```
AIDataValue ( "trackingDistance",   Float,     ( Val = <#> ))
AIDataValue ( "startinactive",      Integer,   ( Val = <#> ))
```

Turrets, whether attached or independent, can orient (aim at targets) but cannot move laterally. In general, turrets can track and fire on targets (although some turrets are not equipped to fire, such as shield generator turrets). A turret has a rest position that it returns to when idle or after collisions.

Two parameters control basic properties of turrets:

- **`trackingDistance`** (floating point; no default) specifies the maximum distance at which the turret tracks targets. Possible targets that are beyond this distance are not tracked. Targets that are tracked are fired on per the turret's fire cone parameters.
- **`startinactive`** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the turret starts inactive when it is first created on a level. Inactive turrets start at their rest position and then transition to the active state per the klown.

Example:

```
AIDataValue ( "trackingDistance",   Float,     ( Val = 1200.0 ))
AIDataValue ( "startinactive",      Integer,   ( Val =    1 ))
```

2.5.1.2 Orbital Turrets (Orbitals)

```
AIDataValue ( "startOrbiting",          Integer,  ( Val = <flag> ))
```

Orbital turrets can be attached to objects via the attachment klown and use the TurretCombat klown for:

- **`startOrbiting`** specifies whether the turret is an orbital (`Val = 1`) or not (`Val = 0`).

Orbital turrets are have their sensing type changed to boring (`kBoringType`) so that they don't show up on sensors separate from their parent. Orbital turrets use dynamic physics.

Note: Orbitals wick electricity by default, but you probably do not want orbitals to take damage from electricity weapons. If you don't want them to wick electricity, use the `doNotWickElectricity` parameter (see Electricity Wicking).

Example:

```
AIDataValue ( "startOrbiting",          Integer,  ( Val = 1 ))
AIDataValue ( "doNotWickElectricity",   Integer,  ( Val = 1 ))
```

2.5.2 Turret Klown

- *Klown Code File:* turret.fsm
- *Klown:* Turret
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, SetTarget, OrientEngines, OrbitalEngines, PollUpdate, FireControl, Weapons, TurretCombat, SearchForTarget, ElectronConductive, Attached, Scorable

The Turret klown uses a set of external klowns to control the functioning of the turret. It has no internal klown. Most weapons-armed turrets attached to objects use the Turret klown.

2.5.3 SphereTurret Klown

- *Klown Code File:* turret.fsm
- *Klown:* SphereTurret
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Collision, SetTarget, PollUpdate, SearchForTarget, Attached, Scorable

The SphereTurretklown uses a set of external klowns to control the functioning of the turret. It has no internal klown. It is used to attach shields to the space stations on Levels 8 and 16.

2.5.4 TractorTurret Klown

- *Klown Code File:* turret.fsm
- *Klown:* TractorTurret
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, SetTarget, OrientEngines, OrbitalEngines, PollUpdate, FireControl, TurretCombat, SearchForTarget, ElectronConductive, TractorBeam2, Scorable
  —*Has InternalKlown:* TractorTurretGuts

The TractorTurretklown uses a set of external klowns to control the functioning of tractor-beam turrets. Its internal klown oversees the tractor beam firing when the turret has a target. (The tractor beam itself is controlled by the TractorBeam2 klown.)

2.5.5 ChargingTurret Klown

- *Klown Code File:* turret.fsm
- *Klown:* Turret
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, SetTarget, OrientEngines, OrbitalEngines, PollUpdate, Weapons, TurretCombat, SearchForTarget, Scorable
  —*Has InternalKlown:* NormalBehavior

The ChargingTurret klown uses a set of external klowns to control the functioning of turrets that can have a maximum number of objects (such as mines) in existance at any one time. Its internal klown manages the turret's firing depending upon whether the maximum number of objects are in existence.

2.5.5.1 Max Shots

```
AIDataValue ( "maxShotFired",      Integer,  ( Val = <#> ))
```

The number of objects the charging turret can have in existence is controlled by:

- **maxShotFired** (integer; default is `kInt32Max`) specifies the number of objects (such as mines) that the turret weapon can have in existence. (Only objects "in play" count—an object that gets destroyed no longer counts.) The weapon fires until the `maxShotFired` limit is reached, whereupon it stops until it is below its limit again.

Example:

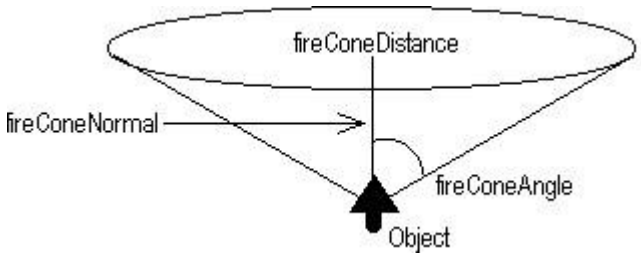
```
AIDataValue ( "maxShotFired",      Integer,  ( Val = 5 ))
```

2.5.5.2 Fire Cone

```
AIDataValue ( "fireConeDistance",   Float,    ( Val = <#> ))
AIDataValue ( "fireConeAngle",      Float,    ( Val = <#> ))
```

A chargin turret can have a fire cone defined for its weapons, which will only fire on targets inside the cone:

- **`fireConeDistance`** (floating point; default is 1200.0) specifies maximum distance (in meters) from the turret that the cone extends.
- **`fireConeAngle`** (floating point; default is 0.95) specifies the cosine of the angle (see Quick Table of Cosines for cosines) used to determine the fire cone (see diagram).



Note: These parameters work identical to the same-named parameters in the FireControl klown. However, they are separate parameters from those in the the FireControl klown. Alos note that the ChargingTurret klown does not use a `fireConeNormal` parameter like the FireControl klown does.

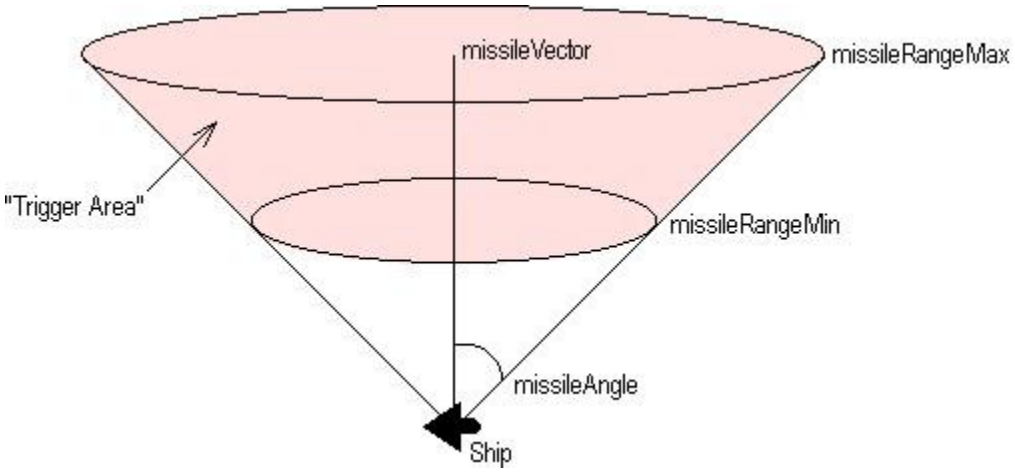
Example:

```
AIDataValue ( "fireConeDistance",   Float,    ( Val = 1200.0 ))
AIDataValue ( "fireConeAngle",      Float,    ( Val =    0.5 ))
```

2.6 Attached Missiles & Mines

- *Klown Code File:* warmissile.fsm
- *Klown:* WarriorMissile
  —*Type:* Submachine

```
AIDataValue ( "missileVector",      Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "missileAngle",       Float,    ( Val = <#> ))
AIDataValue ( "missileRangeMax",    Float,    ( Val = <#> ))
AIDataValue ( "missileRangeMin",    Float,    ( Val = <#> ))
```



An object can have parameters that control missiles and/or mines that are attached to the object:

- **`missileVector`** (vector) specifies the normal direction (in the ship's local coordinates) for the cone. `missileVector` is a normalized (X, Y, Z) vector centered on the ship, so the absolute value of the

components must sum to 1:

(0.0, 1.0, 0.0) is straight in front of the ship.
(0.0, -1.0, 0.0) is directly behind the ship.
(0.0, 0.0, 1.0) is above the ship.
(0.1, 0.2, 0.7) points to the front left top quadrant.

- **missileAngle** (floating point; range is <= +1.0 to >= -1.0) is the cosine (see Quick Table of Cosines) of the angle from the normal direction that is used to define the cone.
- **missileRangeMax** (floating point) is the maximum distance (in meters) of the trigger area for the cone. If the potential target is in the cone but is beyond the `missileRangeMax`, it is not targeted.
- **missileRangeMin** (floating point) is the minimum distance (in meters) of the trigger area for the cone. If the potential target is in the cone but is inside the `missileRangeMax`, it is not targeted.

Example:

```
AIDataValue ( "missileVector",     Vector,   ( Val = ( 0.0, 1.0, 0.0 )))
AIDataValue ( "missileAngle",      Float,    ( Val =    0.0 ))
AIDataValue ( "missileRangeMax",   Float,    ( Val = 2000.0 ))
AIDataValue ( "missileRangeMin",   Float,    ( Val =  750.0 ))

defres rtTurretInfo ( FIGHTER )
(
   TurretPorts[] = ( Name = "left"  ResID = ES_MISSILE1 )
   TurretPorts[] = ( Name = "right" ResID = ES_MISSILE1 )
)
```

2.7 Articulation

2.7.1 Skeleton Klown

- *Klown Code File:* skeleton.fsm
- *Klown:* Skeleton
  —*Type:* Submachine

```
AIDataValue ( "disableSkeletalPhysics", Integer,  ( Val = <#> ))
```

Objects with individually articulated joints use the skeleton klown. Objects with skeletons sometimes need their skeletal physics disabled for certain articulations to work properly (or at all). The following parameter controls this:

- **disableSkeletalPhysics** (integer; default is 0) specifies whether skeletal physics for the ship is enabled (`Val = 0`, the default) or is disabled (`Val = 1`).

Example:

```
AIDataValue ( "disableSkeletalPhysics", Integer, ( Val = 1 ))
```

2.7.2 Common Articulation Parameters

Note: Some parameters in this section reoccur in several klowns spread across different .fsm files (`jointResistance` and `jointDamping`, for example). Since these parameters work in the same way (and almost

always even have the same defaults), they are documented collectively rather than by individual klown. (Note that particular klowns mentioned below can contain any subset of these parameters plus different parameters.)

2.7.2.1 Joint Resistance and Damping

- *Klown Code File:* skeleton.fsm
- *Klown:* Warrior_WavingSpikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

- *Klown Code File:* SpringyThings.fsm
- *Klown:* Exc01_BouncingSpikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton, ExcPain

- *Klown Code File:* wavingSpikes.fsm
- *Klown:* Exc02_Waving Spikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

- *Klown Code File:* whippingtail.fsm
- *Klown:* Exc06_WhippingTail
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

- *Klown Code File:* exc10.fsm
- *Klown:* DEvil_10
  —*Type:* CHFSM
  —*Has Internal Klown:* NormalBehavior
  —*Uses External Klown:* Skeleton (plus others not related to articulation)

```
AIDataValue ( "jointResistance",    Float,    ( Val = <#> ))
AIDataValue ( "jointDamping",       Float,    ( Val = <#> ))
```

Various objects can articulate. Articulation typically requires two basic parameters (although some klowns only need to use one or the other, see below):

- **`jointResistance`** (floating point; default is 200.0 for all above klowns) specifies the stiffness of the movement and can be equal to or greater than 0.0. Values below 10.0 make the movement very loose and values above 90.0 make it very stiff (and at or above 100.0 the joint becomes too stiff to move).

- **`jointDamping`** (floating point; >= 0.0 to <= 1.0; default is 0.9 for all above klowns except Exc01_BouncingSpikes, which defaults to 0.875) specifies the damping factor, which is the time in seconds the articulation movement takes to come to rest.

Who Uses What:

Klown	**`jointResistance`**	**`jointDamping`**
Warrior_WavingSpikes	Yes	Yes
Exc01_BouncingSpikes	No	Yes
Exc02_WavingSpikes	Yes	Yes
Exc06_WhippingTail	Yes	Yes
DEvil_10	Yes	No

Example:

```
AIDataValue ( "jointResistance",  Float, ( Val = 300.0 ))
AIDataValue ( "jointDamping",     Float, ( Val =   0.7 )))
```

2.7.2.2 Pain Reaction

- *Klown Code File:* skeleton.fsm
- *Klown:* ExcPain
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

- *Klown Code File:* skeleton.fsm
- *Klown:* WarriorPain
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton, Health

```
AIDataValue ( "painReaction",       Float,   ( Val = <#> ))
AIDataValue ( "painReactionTime",   Float,   ( Val = <#> ))
```

Note: There are two separate klowns that govern pain reactions: WarriorPain for warriors and ExcPain for bosses. Since both klowns use two parameters that are named the same in each klown, they are documented together. Note, however, that the klowns themselves do not operate identically (although their overall operations are quite similar: to produce visible reactions of pain to damage). Also note that although the parameters' defaults are the same for both klowns, it is possible they could diverge if they are set differently at some future date.

Ships that have skeletons can have articulations that react to pain (damage received by the ship). Reaction to pain is implemented by two parameters (in addition to joint resistance and damping above):

- **painReaction** (floating point; default is 0.0) specifies how much to react under pain; values under 1.0 make the ship articulate negligibly in reaction to pain, while higher values cause more intense articulations. (The pain reaction basically is a random noise-based articulation. When the ship feels pain, the articulations randomly wiggle. For warriors but not bosses, the pain is proportional to amount of health that the warrior has remaining.)

- **painReactionTime** (floating point; default is 0.0) specifies the time (in seconds) the ship spends reacting to the pain.

Example:

```
AIDataValue ( "painReaction",     Float, ( Val =   2.0 ))
AIDataValue ( "painReactionTime", Float, ( Val =   2.0 ))
```

2.7.3 Winged Warriors

- *Klown Code File:* warmisc.fsm
- *Klown:* WarriorWings
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton, PollFast, MatchEngines

```
AIDataValue ( "wingSpan",    Float,   ( Val = <#> ))
AIDataValue ( "wingSpeed",   Float,   ( Val = <#> ))
```

Some ships have wings that can be opened and closed. In attack mode when a ship is rolling its wings flap open, and the wings close when the ship stops rolling. The amount to which the wings open depends upon how much the ship is rolling. Two parameters govern these wing behaviors:

- **wingSpan** (floating point; default is 0.0) specifies the maximum angle (in radians; a radian is approximately 57°17') to which the wings can unfold.
- **wingSpeed** (floating point; default is 0.0) specifies how fast (in radians per second) the wings open and close.

Example:

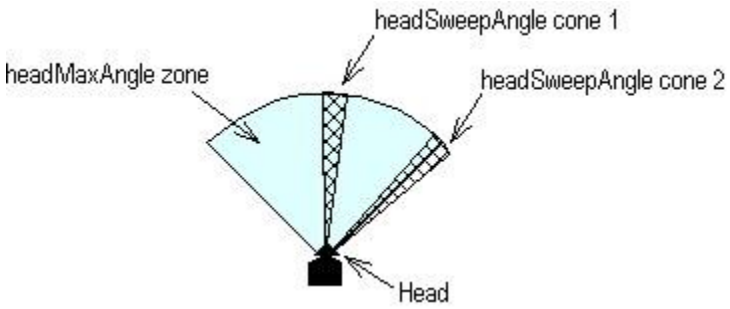
```
AIDataValue ( "wingSpan",    Float,   ( Val = 1.5 ))
AIDataValue ( "wingSpeed",   Float,   ( Val = 1.5 ))
```

2.7.4 Head Sweeping

- *Klown Code File:* wavingSpikes.fsm
- *Klown:* Exc02_Waving Spikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton
- *Klown Code File:* whippingtail.fsm
- *Klown:* Exc06_WhippingTail
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
AIDataValue ( "headMaxAngle",        Float,    ( Val = <#> ))
AIDataValue ( "headSweepAngle",      Float,    ( Val = <#> ))
AIDataValue ( "headSweepSpeed",      Float,    ( Val = <#> ))
AIDataValue ( "headTargetOffset",    Vector,   ( Val = ( <#>, <#>, <#> )))
```

Objects can articulate their heads (and any other subjoint to which this articulation is appropriate), sweeping them and striking targets with them:



- **headMaxAngle** (floating point; >= -1.0 to <= 1.0; default is 0.8) specifies the cosine of the angle (see Quick Table of Cosines) that defines the head's maximum turning zone. The head can turn to "look at" its target only within this zone.
- **headSweepAngle** (floating point; >= -1.0 to <= 1.0; default is 0.2) specifies the cosine of the angle that defines the head's actual sweep cone for a specific sweep. The normal of the cone is whatever direction the head is currently facing in its `headMaxAngle`. While the head can sweep anywhere in its `headMaxAngle` maximum sweep zone, a single sweep can move only in its more limited `headSweepAngle` cone. Note that if the `headSweepAngle` cone extends beyond the `headMaxAngle` (see diagram), the head still will not sweep outside its `headMaxAngle` zone. A head can sweep through the entire volume of Cone 1, while it could only sweep through the part of Cone 2 that lies within the `headMaxAngle` zone. (If the head is turned to its full `headMaxAngle`, then the head can only sweep through half its `headSweepAngle` cone.)
- **headSweepSpeed** (floating point; default is 70.0) specifies the speed of the head's sweep (higher values mean faster sweeps).

- **headTargetOffset** (vector; default is 0.0, 12.0, 0.0) specifies the offset for the head at which it sweeps—you can have the head strike at its target at an offset rather than dead-on.

Example:

```
AIDataValue ( "headMaxAngle",        Float,    ( Val =   0.8 ))
AIDataValue ( "headSweepAngle",      Float,    ( Val =   0.0 ))
AIDataValue ( "headSweepSpeed",      Float,    ( Val = 100.0 ))
AIDataValue ( "headTargetOffset",    Vector,   ( Val = ( 0.0, 0.0, 25.0 )))
```

2.7.5 Striking Arms

- *Klown Code File:* strikingarm.fsm
- *Klown:* StrikingArms
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
AIDataValue ( "armMaxAngle",         Float,    ( Val = <#> ))
AIDataValue ( "armRange",            Float,    ( Val = <#> ))
AIDataValue ( "armSpeed",            Float,    ( Val = <#> ))
AIDataValue ( "armFlex",             Float,    ( Val = <#> ))
AIDataValue ( "armStrikeTime",       Float,    ( Val = <#> ))
AIDataValue ( "armRestTime",         Float,    ( Val = <#> ))
AIDataValue ( "armLeftStrikeAxis",   Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "armRightStrikeAxis",  Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "armSensor",           Resource, ( Val = ( rtBodySensor, <resource ID> )))
```

Bosses can have one or more arms that can strike out and hit the player:

- **armMaxAngle** (floating point; >= -1.0 to <= 1.0; default is 0.5) specifies the cosine of the angle (see Quick Table of Cosines) that defines the arm's sweep zone. The arm moves only within this cone.
- **armRange** (floating point; default is 200.0) specifies the maximum distance (in meters) to the target within which the arm attempts to strike the target. (If the target is beyond the distance, the arm does not strike.)
- **armSpeed** (floating point; default is 200.0) specifies the speed of the arm's strike (higher values mean faster strikes).
- **armFlex** (floating point; >= 0.0 to <= 1.0; default is 0.2) specifies how much the arm flexes in its joint (higher values mean more flex).
- **armStrikeTime** (floating point; default is 1.0) specifies the amount of time (in seconds) the arm attempts to strike its target before it must rest. The arm swings continuously during its `armStrikeTime` (and the number of swings it makes in this time depends upon the other parameters like `armSpeed`.)
- **armRestTime** (floating point; default is 1.0) specifies the amount of time (in seconds) the arm must rest without swinging once its `armStrikeTime` expires.
- **armLeftStrikeAxis** (vector; no default) specifies an offset for the arm so that it can make a "left-handed" swing at the target.
- **armRightStrikeAxis** (vector; no default) specifies an offset for the arm so that it can make a "right-handed" swing at the target.
- **armSensor** (resource; no default) specifies the sensor the striking arm uses to detect targets. This resource must be defined separately. See Sensors for more details on sensors.

Example:

```
AIDataValue ( "armMaxAngle",        Float,    ( Val =   0.5 ))
AIDataValue ( "armRange",           Float,    ( Val = 200.0 ))
AIDataValue ( "armSpeed",           Float,    ( Val = 200.0 ))
AIDataValue ( "armFlex",            Float,    ( Val =   0.2 ))
AIDataValue ( "armStrikeTime",      Float,    ( Val =   1.0 ))
AIDataValue ( "armRestTime",        Float,    ( Val =   1.0 ))
AIDataValue ( "armLeftStrikeAxis",  Vector,   ( Val = (  1.0, 1.0, -0.5 )))
AIDataValue ( "armRightStrikeAxis", Vector,   ( Val = ( -1.0, 1.0, -0.5 )))
AIDataValue ( "armSensor",          Resource, ( Val = ( rtBodySensor, EXC02_ARM_SENSOR )))
```

2.8 Tractor Beams

Various objects (the gate and bosses) use tractor beams, which can tractor in or repel a target caught in the beam.

2.8.1 Tractor Beam 1

- *Klown Code File:* tractorbeam.fsm
- *Klown:* TractorBeam

This tractor beam and its klown are obsolete and are not documented here.

2.8.2 Tractor Beam 2

- *Klown Code File:* tractorbeam2.fsm
- *Klown:* TractorBeam2
  —*Type:* Submachine

```
   // Basic beam resources
AIDataValue ( "tractorBeam",          Integer,  ( Val = <resource ID> ))
AIDataValue ( "beamWhileOnSeq",       Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "beamWhileOffSeq",      Resource, ( Val = ( rtSequence, <resource ID> )))

   // Beam operating parameters
AIDataValue ( "beamPower",             Float,   ( Val = <#> ))
AIDataValue ( "beamMaxVel",            Float,   ( Val = <#> ))
AIDataValue ( "beamTorque",            Float,   ( Val = <#> ))
AIDataValue ( "beamMaxAngVel",         Float,   ( Val = <#> ))
AIDataValue ( "beamRange",             Float,   ( Val = <#> ))
AIDataValue ( "beamInitPower",         Float,   ( Val = <#> ))
AIDataValue ( "beamGain",              Float,   ( Val = <#> ))
AIDataValue ( "beamAttenuateWithDist", Integer, ( Val = <#> ))
AIDataValue ( "beamPortOffset",        Vector,  ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "beamDoubleApproach",    Integer, ( Val = <#> ))
AIDataValue ( "beamNumBeams",          Integer, ( Val = <#> ))
AIDataValue ( "beamArrivedDistance",   Float,   ( Val = <#> ))
AIDataValue ( "beamThrowDist",         Float,   ( Val = <#> ))
AIDataValue ( "maxTractorAngle",       Float,   ( Val = <#> ))
```

```
   // Beam graphical parameters
AIDataValue ( "beamTextureLength",     Float,   ( Val = <#> ))
AIDataValue ( "beamWidth",             Float,   ( Val = <#> ))
AIDataValue ( "beamAlphaOffset",       Float,   ( Val = <#> ))
AIDataValue ( "tractorStartCap",       Integer, ( Val = <resource ID> ))
AIDataValue ( "tractorEndCap",         Integer, ( Val = <resource ID> ))
AIDataValue ( "tractorSphEndCap",      Integer, ( Val = <resource ID> ))
```

2.8.2.1 Basic Beam Resources

- **tractorBeam** (resource ID; no default) specifies the resource ID used for the beam object
- **beamWhileOnSeq** (resource; no default) specifies the `rtSequence` resource that is played when the beam is on.
- **beamWhileOffSeq** (resource; no default) specifies the `rtSequence` resource that is played when the beam is turned off.

2.8.2.2 Beam Operating Parameters

- **beamPower** (floating point; default is 2000.0) specifies the maximum pulling power of the tractor beam.
- **beamMaxVel** (floating point; default is 200.0) specifies the maximum velocity at which the tractor beam pulls its target.
- **beamTorque** (floating point; default is 2000.0) specifies the maximum power the tractor beam can use to turn its target toward the object using the beam.
- **beamMaxAngVel** (floating point; default is 2.0) more or less specifies the maximum angular velocity at which the beam turns its target. (It's not necessarily the maximum, as there're some complicating factors, but bigger values mean faster rotation.)
- **beamRange** (floating point; default is 800.0) specifies the maximum range (in meters) of the tractor beam. The beam cannot grab a target beyond this range.
- **beamInitPower** (floating point; >= 0.0 to <=1.0; default is 0.1) specifies the initial power of the beam. This is a multiplicative factor used to scale the beam's power. (For example, `beamPower` with `Val = 2000.0` and `beamInitPower` with `Val = 0.1` means the beam starts at 200.0 and scales up to 2000.0.) See `beamGain` below for how the beam goes to full power.
- **beamGain** (floating point; >= 0.0 to <=1.0; default is 0.02) specifies beam's power gain percentage per second. For example, `Val = 1.0` means the beam increases to 100% of its full power in one second.
- **beamAttenuateWithDist** (integer; 0 or 1; default is 1) specifies whether (`Val = 1`) or not (`Val = 0`) the beam's power attenuates over distance.
- **beamPortOffset** (vector; default is 0.0, 0.0, 0.0) specifies how much the beam is offset from the port emitting the beam.
- **beamDoubleApproach** (integer; 0 or 1; default is 0) specifies whether or not the takes the facing of its owning object into account. If `Val = 0`, then the beam pulls its target toward the front of the owning object. If `Val = 1`, then the beam pulls its target without regard to the owning object's orientation. (This was used by jumpgates to pull the player into the gate without distinguishing the front of the gate from its back.)
- **beamNumBeams** (integer; default is 1) specifies the number of individual beams that make up the tractor beam.

- **beamArrivedDistance** (floating point; default is -1) specifies the distance (in meters) from the beam's owning object at which the beam ceases to pull its target toward its owning object. The default setting means this functionality is not used.
- **beamThrowDist** (floating point; default is -1) specifies the distance (in meters) from the target's position that the beam throws its target, when the beam is reverse to throw targets. The default setting means this functionality is not used.
- **maxTractorAngle** (floating point; default is 45.0) specifies the angle (in degrees) of the cone in which the tractor beam can operate. This can constrain tractor beams so that they can only be emitted from ships if the target is in front of the ship.

2.8.2.3 Graphical Parameters

- **beamTextureLength** (floating point; default is 300.0) specifies the length of the beam's texture. (This keeps the texture's texels constant.)
- **beamWidth** (floating point; default is 1.0) specifies the width of the beam.
- **beamAlphaOffset** (floating point; default is 0.0) specifies the offset used for the alpha of the beam's texture.
- **tractorStartCap** (resource ID; no default) specifies the resource ID used for the beam's starting cap.
- **tractorEndCap** (resource ID; no default) specifies the resource ID used for the beam's end cap.
- **tractorSphEndCap** (resource ID; no default) specifies the resource ID used for the beam's spherical end cap.

Example:

```
   // Basic beam resources
AIDataValue ( "tractorBeam",          Integer,  ( Val = TRACTOR_BEAM ))
AIDataValue ( "beamWhileOnSeq",       Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "beamWhileOffSeq",      Resource, ( Val = ( rtSequence, <resource ID> )))

   // Beam operating parameters
AIDataValue ( "beamPower",             Float,   ( Val =   2.0e+10 ))
AIDataValue ( "beamMaxVel",            Float,   ( Val = 400.0 ))
AIDataValue ( "beamTorque",            Float,   ( Val =   2.0e+2 ))
AIDataValue ( "beamMaxAngVel",         Float,   ( Val =  15.0 ))
AIDataValue ( "beamRange",             Float,   ( Val = 3000.0 ))
AIDataValue ( "beamInitPower",         Float,   ( Val =    0.0 ))
AIDataValue ( "beamGain",              Float,   ( Val =    0.5 ))
AIDataValue ( "beamAttenuateWithDist", Integer, ( Val =    0 ))
AIDataValue ( "beamPortOffset",        Vector,  ( Val =  ( 0.0, 500.0, -100.0 )))
AIDataValue ( "beamDoubleApproach",    Integer, ( Val =    0 ))
AIDataValue ( "beamNumBeams",          Integer, ( Val =    1 ))
AIDataValue ( "beamArrivedDistance",   Float,   ( Val =  300.0 ))
AIDataValue ( "beamThrowDist",         Float,   ( Val = 1000.0 ))
AIDataValue ( "maxTractorAngle",       Float,   ( Val =   60.0 ))

   // Beam graphical parameters
AIDataValue ( "beamTextureLength",     Float,   ( Val = 1000.0 ))
AIDataValue ( "beamWidth",             Float,   ( Val =    1.0 ))
AIDataValue ( "beamAlphaOffset",       Float,   ( Val =    0.0 ))
AIDataValue ( "tractorStartCap",       Integer, ( Val = GATE_STAR_CAP ))
```

```
AIDataValue ( "tractorEndCap",         Integer, ( Val = GATE_TRACTOR_CAP ))
AIDataValue ( "tractorSphEndCap",      Integer, ( Val = GATE_SPH_END ))
```

2.9 Ship Launching

For `launchList` and `launchDistance`. see `rtLaunchInfo` resources in the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*.)

Example:

```
defres rtLaunchInfo ( LARGESAT2_LAUNCHBAY )
(
   LaunchPorts[] =
   (
      Name          = "pad0"                  // Name of lauch pad joint on model
      ResID         = FIGHTER                 // Name of tick to be attached
      LaunchPoint   = ( 0.0,  700.0, 400.0 )  // Where tick will fly
                                              // after receiving launch instruction
      ApproachPoint = ( 0.0,  500.0,   0.0 )
      LandPoint     = ( 0.0,  100.0,   0.0 )
      Capacity      = 0
      Occupied      = 1
   )
...
)
```

2.10 Shields

Various ships have shields, which are controlled by their own `rtAI` resources. The ShieldGuts klown controls how damage-absorbing shields operate, while the StartsWithShield klown controls damage-invulnerable shields. Both of these klowns are submachines and thus .r files do not call on them directly, but via other klowns. For example, the Shield klown calls on the ShieldGuts klown (and, that is all the Shield klown does).

2.10.1 ShieldGuts Klown

- *Klown Code File:* shield.fsm
- *Klown:* ShieldGuts
  —*Type:* Submachine

```
   // Shield sequences
AIDataValue ( "startSequence",        Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "spangSequence",        Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "splashSpangSequence",  Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "deathSequence",        Resource, ( Val = ( rtSequence, <resource ID> )))

   // Shield operating parameters
AIDataValue ( "ShieldHealth",         Float,    ( Val = <#> ))
AIDataValue ( "ShieldRegenTime",      Float,    ( Val = <#> ))
AIDataValue ( "ShieldDamageFraction", Float,    ( Val = <#> ))
```

```
AIDataValue ( "ShieldVisible",        Integer,  ( Val = <#> ))
```

The ShieldGuts klown controls how damage-absorbing shields operate.

2.10.1.1 Shield Sequences

Every ShieldGuts shield has four sequences:

- **`startSequence`** (resource; no default) specifies the `rtSequence` resource that plays on the shield when it is activated.
- **`spangSequence`** (resource; no default) specifies the `rtSequence` resource that plays on the shield when it is hit.
- **`splashSpangSequence`** (resource; no default) specifies the `rtSequence` resource that plays on the shield when it takes splash damage and emits spangs.
- **`deathSequence`** (resource; no default) specifies the `rtSequence` resource that plays on the shield when it is deactivated (when dies or is otherwise turned off).

Example:

```
AIDataValue ( "startSequence",        Resource, ( Val = ( rtSequence, SEQ_EXC_SHIELD_START
    ) ))
AIDataValue ( "spangSequence",        Resource, ( Val = ( rtSequence, SEQ_EXC_SHIELD_SPANG
    ) ))
AIDataValue ( "splashSpangSequence",  Resource, ( Val = ( rtSequence,
    SEQ_EXC_SHIELD_SPLASH ) ))
AIDataValue ( "deathSequence",        Resource, ( Val = ( rtSequence, SEQ_EXC_SHIELD_DEATH
    ) ))
```

2.10.1.2 Shield Operating Parameters

A shield has a current health value (not exposed in an AI parameter) that tracks its health. When first activated, the shield's current health is set to the shield's maximum health. Thereafter, as the shield takes damage, its health decreases. A shield can regenerate, with its current health regaining strength over time (up to its maximum strength).

A fraction of the damage a shield receives can also be applied to the shield's parent object. (Note that this pass-along damage is applied to both the shield and the parent. For example, if the shield gets hit with 10 damage points and 20% of the damage gets passed along to the parent, the shield takes 10, not eight, points of damage and the parent takes two points.)

A shield's operation is controlled by the following parameters:

- **`ShieldHealth`** (floating point; default is -1.0) specifies the maximum health of the shield. When the shield is activated, it starts at this health. If `ShieldHealth` is set below 0.0 (as its default value is), then the shield's health is not affected by any amount of damage.
- **`ShieldRegenTime`** (floating point; default is 0.0) specifies, if set greater than 0.0, the time (in seconds) it takes for the shield to regenerate half its `ShieldHealth` value. That is, while the shield is active, once every `ShieldRegenTime` seconds, 0.5 * `ShieldHealth` is added to the shield's current health (although current health cannot be increased above the maximum health of the shield). If `ShieldRegenTime` is set to be equal to or less than 0.0, then the shield does not regenerate at all.

- **ShieldDamageFraction** (floating point; default is 0.0) specifies, if set greater than 0.0, the fraction of the damage the shield receives that is also applied to the shield's parent object. (For example, if `Val = 0.5`, then 50% of the damage that strikes the shield is also applied to the parent object.)
- **ShieldVisible** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the shield is visible at all times. When `Val = 0`, the shield is visible only when it is taking damage.

Example:

```
AIDataValue ( "ShieldHealth",          Float,    ( Val = 20.0 ))
AIDataValue ( "ShieldRegenTime",       Float,    ( Val =  2.0 ))
AIDataValue ( "ShieldDamageFraction", Float,    ( Val =  3.3 ))
AIDataValue ( "ShieldVisible",         Integer,  ( Val =  1   ))
```

2.10.2 Shield Klown

- *Klown Code File:* shield.fsm
- *Klown:* Shield
  —*Type:* CHFSM
  —*Has Internal Klown:* ShieldGuts

When all you need is the ShieldGuts functionality, use the Shield klown. This klown simply calls on the ShieldGuts klown.

2.10.3 StartWithShield Klown

- *Klown Code File:* StartWithShield.fsm
- *Klown:* StartWithShield

```
AIDataValue ( "shieldContainer",      Integer,  ( Val = <resource ID> ))
AIDataValue ( "shieldStartsInactive", Integer,  ( Val = <#> ))
AIDataValue ( "shieldGenStartSeq",    Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "shieldTime",           Float,    ( Val = <#> ))
```

The StartWithShield klown controls how damage-invulnerable shields operate. While on, these shields are not affected by damage (and the objects they shield also take no damage). Although any object can in theory use these shields, in the game only a few bosses actually use them. *Note:* Despite the klown name, "StartWithShield," a parameter actually controls whether the shield starts on or off when the object with the shield is created.

To specify the shield an object uses and whether the shield is already active when the object is emitted, set the following parameters:

- **shieldContainer** (integer; no default) specifies the resource ID of the shield container.
- **shieldStartsInactive** (integer; default is 0) specifies whether the shield starts active or inactive when its boss is emitted. `Val = 0` (the default) means the shield is active when the boss is emitted; `Val = 1` means the shield is inactive when the boss is emitted.
- **shieldGenStartSeq** (resource; no default) specifies the `rtSequence` resource that plays when the shield is generated.
- **shieldTime** (floating point; default is 0.0) specifies the time (in seconds) the shield is active once it is generated. *Warning:* Although this is a floating point variable, the klown code onverts it to an integer, rounding it off to the nearest whole number.

Example:

```
AIDataValue ( "shieldContainer",      Integer,  ( Val = EXC_INVUL_SHIELD ))
AIDataValue ( "shieldStartsInactive", Integer,  ( Val = 1 ))
AIDataValue ( "shieldGenStartSeq",    Resource, ( Val = ( rtSequence, EXC_SHIELD_GENERATOR
                ) ))
AIDataValue ( "shieldTime",           Float,    ( Val = 8.0 ))
```

2.10.4 ShieldGenerator Klown

- *Klown Code File:* shield.fsm
- *Klown:* ShieldGenerator
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, PollUpdate, ElectronConductive, StartWithShield, Team
  —*Has Internal Klown:* ShieldGenGuts

The ShieldGenerator klown is used by objects that have shield generators. This klown calls on the StartWithShield klown for its shield effects.

The ShieldGenerator klown has an internal ShieldGenGuts klown, which simply sets the shield generator to be detected as a turret (and thus can be detected separate from its parent object).

2.11 Cloaks

2.11.1 Cloaks are Boring?

- *Klown Code File:* team.fsm
- *Klown:* Team
  —*Type:* Submachine

A cloaked object cannot be "seen" by other objects. This is handled through the Team klown, which changes object's usual sensing type to "boring type" (`kBoringType`; see Sensing Types) when the object's cloak is activated. Sensors do not sense boring types (unless they are specifically set to do so) and thus the object is ignored by the sensors. (Cloaked objects are also visually distinguished, so that players cannot see the objects on their screens.)

When an object's cloak is deactivated, the Team klown changes the object's sensing type back to its normal type.

There are no cloaking parameters exposed as AI data values in the Team klown. See Cloaked Enemies below for the cloaking parameter for cloaked enemies; see The Player Ship for cloaking parameters for the player's cloak special.

Note: Cloaking is handled in the Team klown because once cloaked objects were reassigned to a "cloaked" team.

2.11.2 Cloaked Enemies

- *Klown Code File:* cloak.fsm
- *Klown:* CloakedEnemy
  —*Type:* Submachine
  —*Uses External Klown:* PollUpdate

```
AIDataValue ( "decloakTime",   Float, ( Val = <#> ))
```

It's simple to create a cloaked version of a ship. Take the original container:

```
defres rtTemplate ( TICK1 )
(
   Property( rtGeometry,   TICK1 )
   Property( rtCollision,  TICK1 )
   Property( rtPhysics,    TICK1 )
   Property( rtAI,         TICK1 )
   Property( rtSequence,   SEQ_TICK1_START )
)
```

and make a copy called `CLOAKED_FOO`, with one extra property, an `rtAIOverride`:

```
defres rtTemplate ( CLOAKED_TICK1 )
(
   Property( rtGeometry,   TICK1 )
   Property( rtCollision,  TICK1 )
   Property( rtPhysics,    TICK1 )
   Property( rtAI,         TICK1 )
   Property( rtSequence,   SEQ_TICK1_START )
   Property( rtAIOverride, CLOAKED_TICK1 )
)
```

The parameter is:

- **decloakTime** (floating point; default it 0.0) the specifies the time (in seconds) a ship's cloak is turned off when the ship fires a weapon or when it gets hit by a shot.

Example:

```
defres rtAIOverride( CLOAKED_TICK1 )
(
   Type.Klown = ( KlownName = "Warrior" )
   AIDataValue ( "decloakTime",   Float, ( Val = 1.0 ))
)
```

2.11.3 Energy-Consuming Cloaks

- *Klown Code File:* cloak.fsm
- *Klown:* Cloak
  —*Type:* Submachine
  —*Uses External Klowns:* PollUpdate, ShipModel

```
AIDataValue ( "CloakMaxEnergy",             Float, ( Val = <#> ))
AIDataValue ( "EnergyConsumptionPerTick",   Float, ( Val = <#> ))
AIDataValue ( "CloakAbortEnergy",           Float, ( Val = <#> ))
AIDataValue ( "CloakMinEnergy",             Float, ( Val = <#> ))
```

When the klown is operating, it runs a cloak that consumes energy each tick (0.05 seconds) while activated. If the cloak falls below its abort energy level (or the cloaked object takes damage), the cloak is deactivated. While deactivated, the cloak recharges its energy each tick, until it is recharged past a minimum energy level, whereupon it is reactivated. The parameters that control this are:

- **CloakMaxEnergy** (floating point; default it 100.0) is the cloak's maximum possible energy and also its starting energy when the Cloak klown begins operating.
- **EnergyConsumptionPerTick** (floating point; default it 1.0) is the amount of energy subtracted from the cloak's current energy level while the cloak is activated. It is also the amount of energy added to the cloak's current energy level while the cloak is deactivated

- **CloakAbortEnergy** (floating point; default it 10.0) is the energy level at which the cloak automatically deactivates.
- **CloakMinEnergy** (floating point; default it 30.0) is the energy level at which a deactivated, recharging cloak reactivates.

Notes: 1) The Cloak klown also calls on the ShipModel cloak, so that you can specify model-related cloaking parameters (`cloakModel`, `cloakFadeTime`). See Scorable Klown. 2) This new organization has rendered the `ContainerModel` parameter obsolete. (`ContainerModel` specified the resource ID of the boss's container and was used for cloaking purposes.)

Example:

```
AIDataValue ( "CloakMaxEnergy",             Float, ( Val = 100.0 ))
AIDataValue ( "EnergyConsumptionPerTick",   Float, ( Val =   1.0 ))
AIDataValue ( "CloakAbortEnergy",           Float, ( Val =  10.0 ))
AIDataValue ( "CloakMinEnergy",             Float, ( Val =  30.0 ))
```

2.12 Teleportation

- *Klown Code File:* Teleport.fsm
- *Klown:* Teleport
  —*Type:* Submachine

2.12.1 Teleportation Sequences

```
AIDataValue ( "FadeOut",   Resource,   ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "FadeIn",    Resource,   ( Val = ( rtSequence, <resource ID> )))
```

The following parameters point to the teleportation sequences:

- **FadeOut** (resource; no default) specifies the `rtSequence` resource that plays when the ship begins teleporting.
- **FadeIn** (resource; no default) specifies the `rtSequence` resource that plays when the ship finishes teleporting.

Example:

```
AIDataValue ( "FadeOut",   Resource,   ( Val = ( rtSequence, EXC11_SEQ_TELEPORT_FADEOUT
                )))
AIDataValue ( "FadeIn",    Resource,   ( Val = ( rtSequence, EXC11_SEQ_TELEPORT_FADEIN )))
```

2.12.2 Teleportation Operating Parameters

```
AIDataValue ( "TeleportDistanceToTarget",   Float,   ( Val = <#> ))
AIDataValue ( "TeleportMaxEnergy",          Float,   ( Val = <#> ))
AIDataValue ( "DamageLevelToAbort",         Float,   ( Val = <#> ))
AIDataValue ( "TeleportMinEnergy",          Float,   ( Val = <#> ))
```

When the klown is operating, it runs a teleporter that consumes energy when it is used. (The teleporter's current energy level is halved whenever the ship successfully teleports or has its teleportation aborted due to damage.) If the teleporter

energy falls below a minimum energy level, the teleporter is deactivated. While deactivated, the teleporter recharges its energy at a rate of one unit of energy per 0.1 second, until it is recharged past its minimum energy level, whereupon it is reactivated. The parameters that control this are:

- **TeleportDistanceToTarget** (floating point; default it 700.0) specifies the distance (in meters) that the teleporting ship reappears in front of its target.
- **TeleportMaxEnergy** (floating point; default it 100.0) is the teleporter's maximum possible energy and also its starting energy when the Teleport klown begins operating.
- **TeleportMinEnergy** (floating point; default it 10.0) is the energy level below which the teleporter is deactivated.
- **DamageLevelToAbort** (floating point; default it 10.0) is the minimum amount of damage that causes a teleporter to abort. If the teleporter is actively preparing to teleport, the teleportation is aborted (and the current teleport energy level is halved) if the ship takes damage equal to or greater than `DamageLevelToAbort`.

Example:

```
AIDataValue ( "TeleportDistanceToTarget",   Float,   ( Val = 700.0 ))
AIDataValue ( "TeleportMaxEnergy",          Float,   ( Val = 100.0 ))
AIDataValue ( "DamageLevelToAbort",         Float,   ( Val =  10.0 ))
AIDataValue ( "TeleportMinEnergy",          Float,   ( Val =  10.0 ))
```

3 Weapons, Shots, and Firing

3.1 Available & Default Weapons

3.1.1 Available Weapons

- *Klown Code File:* none
- *Klown:* none (`AvailWeapons` appears in weapon.cpp and #weapon.cpp# rather than in a klown)

```
AIDataValue ( "AvailWeapons", Resource, ( Val = ( rtGunList, <resource ID> )))
```

An object can have a parameter that points to an `rtGunList` resource listing the weapons for the object:

- **AvailWeapons** (resource; no default) specifies the `rtGunList` resource that contains a list of the object's weapons.

Example:

```
AIDataValue ( "AvailWeapons", Resource, ( Val = ( rtGunList, EXCESSION_01 )))

defres rtGunList ( EXCESSION_01 )
(
   Guns[] =
   (
      Category  = GUNCAT_MAIN
      StartWith = EXW_GREENPLASMA
      Enabled   = 1
   )
   Guns[] =
   (
      Category  = GUNCAT_ANTIMISSILE
      StartWith = EXW_MULTISR
      Enabled   = 1
   )
)
```

3.1.2 Weapons Klown

- *Klown Code File:* weapons.fsm
- *Klown:* Weapons
  —*Type:* Submachine
  —*Has External Klown:* Team

3.1.2.1 Default Weapon

```
AIDataValue ( "DefaultWeapon",       Integer,  ( Val = <resource ID> ))
```

An object can have a default weapon category set:

- **DefaultWeapon** (resource ID; default is -1) specifies the weapon category resource ID for the object's default weapon. The default value specifies that the object has no default weapon category.

Example:

```
AIDataValue ( "DefaultWeapon",        Integer,   ( Val = GUNCAT_MAIN ))
```

3.1.2.2 Auto-Aiming

```
AIDataValue ( "autoAimAngle",         Float,     ( Val = <#> ))
```

Any weapon that uses the Weapons klown (i.e., various enemy weapons) can have auto-aiming. When an auto-aiming weapon fires on a target that is within the auto-aiming cone, the shot auto-aims using leading to try to hit the target. The following parameter defines the auto-aiming cone:

- **autoAimAngle** (floating point; default is 0.0) specifies the angle (in degrees) of the auto-aiming cone.

Example:

```
AIDataValue ( "autoAimAngle",         Float,     ( Val = 180.0 ))
```

3.1.2.3 Continuous Sound for Timed Weapons

```
AIDataValue ( "ContinuousSeq",         Resource,     ( Val = ( rtSequence, <resource ID> )))
```

Timed weapons can have a continuos sound sequence. (This is useful for playing sounds with weapons that have rapid-fire shots.) To connect the sequence to a weapon, use:

the `ContinuousSeq` AIDataValue. It is a sequence which will start being played when firing begins, and will be stopped when firing stops:

- **ContinuousSeq** (resource; no default) the sequence that starts being played when firing begins and stops when firing stops.

Example:

```
defres rtAI (PW_LASER1)
(
   Type.Klown = ( KlownName = "TimedWeapon" )
   AIDataValue ( "Category",         Integer,     ( Val  = GUNCAT_LASER ))
   AIDataValue ( "GunportNames",     Resource,    ( Val  = ( rtGunportNames, PW_LASER1 )))
   AIDataValue ( "FireSeq",          Resource,    ( Val  = ( rtSequence, PW_LASER1_SEQ1 )))
   AIDataValue ( "ContinuousSeq",    Resource,    ( Val  = ( rtSequence, PW_LASER1_SEQ2 )))
   AIDataValue ( "Delay",            Float,       ( Val  = 0.12 ))
   AIDataValue ( "shotSpeed",        Float,       ( Val  = 1200 ))
)
```

3.1.2.4 Suppressing Friendly Fire

AI-controlled objects do not fire if there is a friendly object in the first third of its fire cone. This reduces (but intentionally does not element) friendly fire accidents. If an object has a `safetySensor`, it uses that sensor to detect for friendlies. If it doesn't have a `safetySensor`, it instead uses its `enemySensor` to find friendlies. See Sensors for more details on sensors.

An object checks for the presence of friendlies in its fire cone about once per second.

3.2 Targeting

The basic difference between shots and missiles are that shots are unguided while missiles are self-guided, using engines to correct their courses towards their targets.

3.2.1 SetTarget Klown

- *Klown Code File:* target.fsm
- *Klown:* SetTarget
  —*Type:* Submachine

```
AIDataValue ( "RejectExternalTargets",   Integer,   ( Val = <#> ))
AIDataValue ( "AcceptableTargets",       Integer,   ( Val = <#> ))
AIDataValue ( "targetMinTimeout",        Float,     ( Val = <#> ))
AIDataValue ( "targetMaxTimeout",        Float,     ( Val = <#> ))
```

Objects like guided missiles can use the SetTarget klown to retarget:

- **`RejectExternalTargets`** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the missile will retarget if it already has a target. When `Val = 1`, the SetTarget klown will not retarget the missile to another target as long as it already has a target.
- **`AcceptableTargets`** (integer; default is `kSenseAll`) specifies the acceptable targets of the missile. (This can be use, for example, to have Sinibombs target only bosses and gates.) Use the sensing type variable names (see Sensing Types) to specify the target. To specify multiple different types of targets, OR the types together: `Val = <object type> | <object type> | ....`

All objects periodically lose their current target and acquire a new target. (This is needed for the multiplayer version. For single player play, it is now possible, when an enemy is chasing the player, for the enemy "lose" the player if the player gets far enough away, something which was not possible before without cloaking.) Two parameters control this:

- **`targetMinTimeout`** (floating point; default is 4.0) specifies the minimum time (in seconds) that must pass before targeting times out and forces the object to reacquire a new target.
- **`targetMaxTimeout`** (floating point; default is 20.0) specifies the maximum time (in seconds) that must pass before targeting times out and forces the object to reacquire a new target.

If you want an object to do little or no retargeting, set the minimum and maximum values to very high numbers.

Example:

```
AIDataValue ( "RejectExternalTargets",   Integer,   ( Val = 1 ))
AIDataValue ( "AcceptableTargets",       Integer,   ( Val = kSenseEgg | kSenseMonster ))
AIDataValue ( "targetMinTimeout",        Float,     ( Val =  4.0 ))
AIDataValue ( "targetMaxTimeout",        Float,     ( Val = 10.0 ))
```

3.2.2 SearchForTarget Klown

- *Klown Code File:* target.fsm

- *Klown:* SearchForTarget
  —*Type:* Submachine
  —*Has External Klown:* PollUpdate, Team, SetTarget

```
AIDataValue ( "canTargetTeammates", Integer,  ( Val = <#> ))
```

The SearchForTarget klown uses an algorithm to search for all potential targets within sensing range and to select one as the target.

The klown can be told to target teammates via:

- **canTargetTeammates** (integer; default is 0) specifies whether (`Val = 1` or any integer greater than 1) or not (`Val = 0`) the object can target teammates.

Example:

```
AIDataValue ( "canTargetTeammates", Integer,  ( Val = 1 ))
```

3.3 Shot Klown

- *Klown Code File:* shot.fsm
- *Klown:* Shot
  —*Type:* CHFSM
  Uses External Klowns: Birth, Fader, Health, ScoreVector
  Has Internal Klown: ShotGuts

Shots are unguided energy blasts or beams emitted by various weapons. Shots use sensors,

As a shot flies through space, it checks for collisions just like any other object. A shot always ignores collisions with crystals (`kCrystalType`), power-ups (`kPowerupType`), and other shots (`kShotType`). A shot will also ignore collisions with Sinibombs, unless the `antiSinibomb` is set to have it affect Sinibombs. A shot will collide with all other objects. When it collides with an object, it takes effect (does damage and/or play a sequence on the object) if the object is an acceptable target of the object that fired the shot; otherwise, it riccochets off the object.

3.3.1 Shot Basics

```
AIDataValue ( "damage",          Integer,     ( Val = <#> ))
AIDataValue ( "RichochetSeq",    Resource,    ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "antiSinibomb",    Integer,     ( Val = <#> ))
```

Three parameters define the basic characteristics of shots:

- **damage** (integer; no default) specifies the damage the shots inflicts when it strikes a target other than a Sinistar. (Sinistar are immune from damage from shots.)
- **RichochetSeq** (resource; no default) specifies the sequence that plays when a shot ricochets off an object.
- **antiSinibombS** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the shot can affect Sinibombs. *Note:* For the shot to affect Sinibombs, the shot's sensor must be set to detect missiles. If `antiSinibomb` is used but missiles aren't detected, then Sinibombs will still be unaffected. See Sensors for more details on sensors.

Example:

```
AIDataValue ( "damage",          Integer,     ( Val = 3 ))
```

```
AIDataValue ( "RichochetSeq",   Resource,   ( Val = ( rtSequence, EXAMPLE_BOUNCE_OFF )))
AIDataValue ( "antiSinibomb",   Integer,    ( Val = 1 ))
```

3.3.2 Playing Sequences on Targets

```
AIDataValue ( "acceptableRNATargets", Integer,  ( Val = <#> ))
AIDataValue ( "RNASequence",          Resource, ( Val = ( rtSequence, <resource ID> )))
```

When a shot hits an appropriate target, a sequence can be played on the target. This can be used to play sounds on the target when the shot hits, to change the target in some manner (this is how one boss converts workers into warriors), or anything else that can be done via sequences (see the separate document, The Sequencing System). The parameters are:

- **acceptableRNATargets** (integer; no default) specifies all valid targets on which the `RNASequence` is played if the shot hits the target. Use the sensing types names (see Sensing Types) to specify the targets. (For example, use `kSenseWorker` to specify workers as valid targets.) To specify multiple different types of targets, OR the types together: `Val = <object type> | <object type> | ....`
- **RNASequence** (resource; no default) plays an `rtSequence` resource named *<resource ID>* on the target.

Note that an `RNASequence` can be specified for ***any shot***. We can do some pretty funky stuff with this parameter!

Example:

```
AIDataValue ( "acceptableRNATargets", Integer,  ( Val = kSenseWorker ))
AIDataValue ( "RNASequence",          Resource, ( Val = ( rtSequence,
                  EXS_EGGFROMWORKERSHOT_SEQ1 )))
```

3.4 Missiles & Mines

Missiles and mines are physical objects in the game. They are self-guided, using engines to correct their courses towards their targets.

3.4.1 Guided Missiles

3.4.1.1 Long-Range Missiles (Missile Klown)

- *Klown Code File:* missile.fsm
- *Klown:* Missile
  —*Type:* CHFSM
  —*Uses External Klowns:* SetTarget, SearchForTarget, MissileEngines, Health, Fader, ScoreVector, MissileGuts, ElectronConductive, Scorable

The Missile klown controls how long-range missiles operate. It is set up as a klown over a set of external klowns and has no internal klown of its own. Its MissileGuts klown actually controls the behavior of the missiles.

3.4.1.2 Short-Range Missiles (SRMissile Klown)

- *Klown Code File:* missile.fsm

- *Klown:* SRMissile
  —*Type:* CHFSM
  —*Uses External Klowns:* SetTarget, SearchForTarget, SRMissileEngines, Health, Fader, ScoreVector, MissileGuts, ElectronConductive, Scorable

The SRMissile klown controls how short-range missiles operate. It is set up as a klown over a set of external klowns and has no internal klown of its own. (TheSRMissile klown uses a different engine klown for its missiles than the Missile klown uses.) Its MissileGuts klown actually controls the behavior of the missiles.

3.4.1.3 MissilesGuts Klown

- *Klown Code File:* missile.fsm
- *Klown:* MissileGuts
  —*Type:* Submachine

```
AIDataValue ( "damage",            Integer,    ( Val = <#> ))
AIDataValue ( "excessionDamage",   Integer,    ( Val = <#> ))
AIDataValue ( "launchSequence",    Resource,   ( Val = ( rtSequence,   <resource ID> )))
AIDataValue ( "slowAngle",         Float,      ( Val = <#> ))
AIDataValue ( "Effortdelay",       Float,      ( Val = <#> ))
```

The lifecycle of a missile is:

- *PreLaunch:* If the missile is present in the game before it is launched (for example, when mounted on a ship), it is tagged as `kBoringType` so that sensors will not detect it as an active missile.
- *Launch*: When launched, the missile is tagged as a missle (`kMissileType`) and its launch sequence starts playing. If a weapon launches multiple weapons in one volley, all but the first missile briefly delay—they fly at 75% maximum speed for a set period of time. (This allows for the first missile of the volley to strike a target and the remaining missiles to retarget on other targets.)
- *Targeting:* The missile tries to acquire a target. If no target is available, the missile will flight in a staight path and keep trying to acquire a target during its existence.
- *Maneuvering:* When the missile acquires a target, it uses its engines to manuever to try to strike the target.
- *Collision:* If the missile strikes it target or collides with another object, it performs a series of tests to determine what it does. All missiles ignore collisions with crystals and power-ups. Sinibomb missiles ignore offerings and all shots except anti-Sinibomb shots.
- *Explosion:* If the missile doesn't ignore the collision, it explodes. All missiles can damage non-Sinistar targets. Sinibomb missiles can also damage the Sinistar.
- *Miss:* If a missile reaches the bitter end of its flight without colliding with a valid object, it dies, sad and dejected, cursing a cruel universe that prevented it from fulfilling its life's mission.

The MissileGuts klown uses the followng tunable parameters:

- **`damage`** (integer; default is 0) specifies the damage the missile does to a target other than a Sinistar. (The Sinistar are immune from this type of damage.)
- **`excessionDamage`** (integer; default is 0) specifies the damage the missile does to a Sinistar. A missile with an `excessionDamage` value greater than 0 is a Sinibomb and is the only thing that can damage a Sinistar.
- **`launchSequence`** (resource; no default) specifies the `rtSequence` resource that plays on the missile when it is launched. The sequence can be used for things like smoke trails coming off the missiles.
- **`slowAngle`** (floating point; default is 0.08) specifies how sharp a turn the missile can make—smaller numbers mean sharper turns, but numbers too small mean the missile goes out of control.

- **Effortdelay** (floating point; default is 0.0) specifies the time (in seconds) that all but the first missile missile in a multi-missile volley fly at 75% maximum speed.

Example:

```
AIDataValue ( "damage",            Integer,  (Val = 12   ))
AIDataValue ( "excessionDamage",   Integer,  (Val =  0   ))
AIDataValue ( "launchSequence",    Resource, (Val = ( rtSequence, SEQ_MISSILE_EXAMPLE )))
AIDataValue ( "slowAngle",         Float,    (Val = 0.09 ))
AIDataValue ( "Effortdelay",       Float,    (Val = 2.0  ))
```

3.4.2 Seeking Mines

3.4.2.1 SeekingMine Klown

- *Klown Code File:* mine.fsm
- *Klown:* SeekingMine
  —*Type:* CHFSM
  —*Uses External Klowns:* Health, MissileEngines, OrientEngines, SetTarget, SearchForTarget, Avoidance, ScoreVector, Scorable, Team, SeekingMineGuts

The SeekingMine klown is organized as just a set of external klowns, with no other functionality. The SeekingMineGuts klown below controls the functionality of the mine.

3.4.2.2 SeekingMineGuts Klown

- *Klown Code File:* mine.fsm
- *Klown:* SeekingMineGuts
  —*Type:* Submachine

```
AIDataValue ( "damage",           Integer,    ( Val = <#> ))
AIDataValue ( "launchSequence",   Resource,   ( Val = ( rtSequence,   <resource ID> )))
AIDataValue ( "triggerSequence",  Resource,   ( Val = ( rtSequence,   <resource ID> )))
AIDataValue ( "homeRadius",       Float,      ( Val = <#> ))
AIDataValue ( "triggerDelay",     Float,      ( Val = <#> ))
```

The SeekingMineGuts klown is a submachine used to control the functionality of seeking mines. A seeking mine explodes when it collides with an object that is not on its team and that is not a shot or a missile.

- **damage** (integer; no default) specifies the damage the mine does to a target other than a Sinistar. (The Sinistar are immune from mine damage.)
- **launchSequence** (resource; no default) specifies the `rtSequence` resource that plays when the mine activates.
- **triggerSequence** (resource; no default) specifies the `rtSequence` resource that plays when the mine is triggered.
- **homeRadius** (floating point; default is 0.0) specifies the distance (in meters) to the player beyond which the mine will give up chasing the player and return to its home (starting) position.
- **triggerDelay** (floating point; no default) specifies the time (in seconds) after the mine is created (or launched) that the mine waits before it activates.

Example:

```
AIDataValue ( "damage",         Integer,  ( Val = 30 ))
AIDataValue ( "launchSequence", Resource, ( Val = ( rtSequence, SEQ_DMINES_LAUNCH_SEQ )))
AIDataValue ( "triggerDelay",   Float,    ( Val = 6.0 ))
AIDataValue ( "homeRadius",     Float,    ( Val = 500.0 ))
```

3.5 FireControl Klown

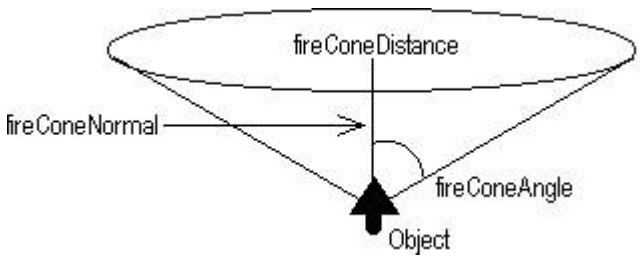
- *Klown Code File:* firecontrol.fsm
- *Klown:* FireControl
  —*Type:* Submachine
  —*Has External Klown:* PollUpdate, FireControlRoot, SetTarget

3.5.1 Fire Cones

```
AIDataValue ( "fireConeNormal",     Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "fireConeDistance",   Float,    ( Val = <#> ))
AIDataValue ( "fireConeAngle",      Float,    ( Val = <#> ))
```

An object can have a fire cone defined for its weapons, which will only fire on targets inside the cone:

- **`fireConeNormal`** (vector; default is 0.0, 0.0, 0.0) specifies the direction of the firing cone (in the object's local coordinates). The default value is the ship's default normal vector.
- **`fireConeDistance`** (floating point; no default) specifies maximum distance (in meters) from the ship that the cone extends.
- **`fireConeAngle`** (floating point; no default) specifies the cosine of the angle (see Quick Table of Cosines if you need to brush up on cosines) used to determine the fire cone (see diagram).



Example:

```
AIDataValue ( "fireConeNormal",     Vector,   ( Val = ( 0.0, 1.0, 0.0 )))
AIDataValue ( "fireConeDistance",   Float,    ( Val = 1200.0 ))
AIDataValue ( "fireConeAngle",      Float,    ( Val =    0.5 ))
```

3.5.2 Overheating Weapons

```
AIDataValue ( "weaponOverheatTime",   Float,    ( Val = <#> ))
AIDataValue ( "weaponCoolingTime",    Float,    ( Val = <#> ))
```

A weapon can overheat when firing and must cool down before firing again:

- **`weaponOverheatTime`** (floating point; no default) specifies time (in seconds) that the weapon can fire before overheating. When it reaches its `weaponOverheatTime`, it must cool down before it can fire again.
- **`weaponCoolingTime`** (floating point; no default) specifies time (in seconds) that an overheated weapon must spend to cool down before it can fire again.

Note: Both `weaponOverheatTime` and `weaponCoolingTime` must be specified for the weapon to use this behavior. If either or both are not specified, then the weapon cannot overheat.

Example:

```
AIDataValue ( "weaponOverheatTime",   Float,    ( Val = 1.6 ))
AIDataValue ( "weaponCoolingTime",    Float,    ( Val = 0.8 ))
```

3.5.3 Experimental: Performance and High-Shot Ships

Experimental: There's an optimization designed to suppress enemy fire when there's too many shots in the air. The affected files are:

Src\applib\cz\klown\submachine
    Firecontrol.fsm
Src\app\cz
    BodySensor.cpp
Dat\app\cz
    tickid.rh
Dat\app\cz\tick
    tick.r etc.
Dat\app\cz\turret
    turret.r etc.

(grep for OptimizeSensor in .r files)

It has not been determined whether this test is useful in the wake of the improved firing algorithm that reduces friendly fire hits. To fully make it work, you'll need to add `optimizeSensor` to all ships that fire a large number of shots. The fire control code checks the `optimizeSensor` when it wants to fire and suppresses its fire for a second if it has detected too much crap in the air.

3.6 FireControlArray Klown: Multiple Fire Cones

- *Klown Code File:* firecontrol.fsm
- *Klown:* FireControlArray
  —*Type:* Submachine
  —*Has External Klown:* PollUpdate, FireControlRoot

```
AIDataValue ( "fireControlInfo", Resource, ( Val = ( rtFireControlInfo, <resource ID> )))
```

The above fire cone parameters specify a singe fire cone for an object. `rtFireControl` resources allow objects with multiple weapons to assign each weapon its own fire cone and its own overheating and cooling time. In the object's AI, use the following parameter to points to a separate `rtFireControlInfo` resource for the object:

- **`fireControlInfo`** (resource; no default) specifies an `rtFireControlInfo` resource. This resource contains a fire control list, with fire control specifications for the categories of the object's weapons.

Example:

```
AIDataValue ( "fireControlInfo", Resource, ( Val = ( rtFireControlInfo, EXCESSION_02 )))

defres rtFireControlInfo( EXCESSION_02 )
(
   FireList[] =
   (
```

```
      Weapon       = GUNCAT_MAIN
      ConeDistance = 1000
      ConeAngle    = 0.8
      OverheatTime = 10.0
      CoolingTime  = 5.0
   )

   FireList[] =
   (
      Weapon       = GUNCAT_MISSILE
      ConeDistance = 2000
      ConeAngle    = 0.8
      OverheatTime = 2.0
      CoolingTime  = 10.0
   )
)
```

3.7 Concussion Klown

- *Klown Code File:* concussion.fsm
- *Klown:* Concussion
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, ConcussionGuts, ScoreVector

```
AIDataValue ( "damage",          Integer,    ( Val = <#> ))
AIDataValue ( "lifetime",        Float,      ( Val = <#> ))
AIDataValue ( "enemySensor",     Resource,   ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "spangSequence",   Resource,   ( Val = ( rtSequence,   <resource ID> )))
AIDataValue ( "antiSinibomb",    Integer,    ( Val = <#> ))
```

The Concussion klown through its ConcussionGuts submachine controls concussion shots:

- **lifetime** (floating point; default is -1.0) specifies the lifetime of the concussion shot. A concussion shot automatically detonates at the end of its lifetime if it hasn't previously detected a target and detonated. If lifetime is set to a negative number, then the concussion shot does not have a lifetime.
- **damage** (integer; no default) specifies the maximum damage the concussion shot does at the center of its blast. Damage falls off over distance. (The Sinistar are immune from this damage.)
- **enemySensor** (resource; no default) specifies the sensor the concussion shot uses to detect objects that will cause the shot to detonate. This resource must be defined separately. See Sensors for more details on sensors.
- **spangSequence** (resource; no default) specifies the `rtSequence` resource that plays when the shot detonates.
- **antiSinibombS** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the concussion shot can affect Sinibombs. *Note:* For the shot to affect Sinibombs, the shot's sensor must be set to detect missiles. If `antiSinibomb` is used but missiles aren't detected, then Sinibombs will still be unaffected. See Sensors for more details on sensors.

Note: Various aspects of how the concussion blast itself works are controlled by blast wave parameters.

Example:

```
AIDataValue ( "damage",          Integer,    ( Val = 24   ))
```

```
AIDataValue ( "lifetime",         Float,      ( Val = 1.66 ))
AIDataValue ( "enemySensor",      Resource,   ( Val = ( rtBodySensor, PS_BURST1      )))
AIDataValue ( "spangSequence",    Resource,   ( Val = ( rtSequence,   PS_BURST1_SEQ1 )))
AIDataValue ( "antiSinibomb",     Integer,    ( Val = 1 ))
```

4 Boss Behaviors

Boss have common behaviors, which are behaviors that all bosses theoretically can use (although in practice some do not use some of these), and specific behaviors, which apply only to a particular boss.

4.1 Common Boss Behaviors

These behaviors are shared by several bosses.

In addition to the tunable behaviors listed below, bosses have some common, hardcoded behaviors:

- **Effort Levels:** All bosses can have effort levels, which can influence common or specific behaviors. A boss's effort level for a behavior is a multiplier to the boss's normal `rtBodyMotion` and `rtBodyControl` parameters. For example, a boss with an effort level of 2.0 for a behavior has its `rtBodyMotion` and `rtBodyControl` parameters doubled for that behavior. Effort levels are currently hardwired into the bosses and cannot be set via .r files. If a boss's effort level is not listed for a behavior, its effort level is 1.0. When a boss ends a behavior at which it has its effort level set to a value other than 1.0, it has its effort level reset to 1.0 just before it begins its next behavior. (The next behavior would then set the effort level as appropriate for that behavior.)
- **Peace States:** A boss's behaviors almost always depend on whether or not the boss has detected an enemy (the player). If it has not detected an enemy, the boss enters its peace state behavior. This can vary from boss to boss but typically consists of the boss hanging out near its home position and trying to detect an enemy. Peace states are hardcoded.

4.1.1 ExcSound Klown: Boss Sounds

- *Klown Code File:* excsound.fsm
- *Klown:* ExcSound
  —*Type:* Submachine

4.1.1.1 Birth, Death, and Ambient Sounds

```
AIDataValue ( "BirthSound", Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "DeathSound", Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "ShipSound",  Resource, ( Val = ( rtSequence, <resource ID> )))
```

Every boss has three well-defined sounds: its birth sound, its death sound, and its ambient ship sound. These are played in sequences pointed to by three parameters:

- **`BirthSound`** specifies the `rtSequence` resource that is played when the boss is born (arrives on a level).
- **`DeathSound`** specifies the `rtSequence` resource that is played when the boss is killed.
- **`ShipSound`** specifies the `rtSequence` resource that is played while the boss is present on a level.

Example:

```
AIDataValue ( "BirthSound", Resource, ( Val = ( rtSequence, EXC01_SEQ_SND_BIRTH )))
```

```
AIDataValue ( "DeathSound", Resource, ( Val = ( rtSequence, EXC01_SEQ_SND_DEATH )))
AIDataValue ( "ShipSound",  Resource, ( Val = ( rtSequence, EXC01_SEQ_SND_SHIP  )))
```

4.1.1.2 Roars, Screams, Pain Sounds, and Flee Taunts

```
AIDataValue ( "roarSeqs",      Resource, ( Val = ( rtIntegerArray, <resource ID> )))
AIDataValue ( "painSeqs",      Resource, ( Val = ( rtIntegerArray, <resource ID> )))
AIDataValue ( "screamSeqs",    Resource, ( Val = ( rtIntegerArray, <resource ID> )))
AIDataValue ( "fleeTauntSeqs", Resource, ( Val = ( rtIntegerArray, <resource ID> )))
```

Every boss also has four other sounds: its roars, screams, pain sounds, and flee taunts. Roars happen randomly if no other boss sounds were played. Flee taunts play on certain bosses as they are about to charge. Screams play on certain bosses as they try to ram the player. For variety, it is possible to specify a number of different sounds for each of these boss sound categories. The following parameters control these categories:

- **roarSeqs** specifies the `rtIntegerArray` resource that contains the list of the boss's roar sequences. (See below for how `rtIntegerArray` resources are used for boss sounds.) Roars are played at random times if no other sounds have been played for a certain amount of time.
- **painSeqs** specifies the `rtIntegerArray` resource that contains the list of the boss's pain sound sequences. (See below for how `rtIntegerArray` resources are used for boss sounds.) Pain sounds are played when the boss is hit with a Sinibomb.
- **screamSeqs** specifies the `rtIntegerArray` resource that contains the list of the boss's scream sequences. (See below for how `rtIntegerArray` resources are used for boss sounds.) Screams are played when the bosses start an attack.
- **fleeTauntSeqs** specifies the `rtIntegerArray` resource that contains the list of the boss's flee taunt sequences. (See below for how `rtIntegerArray` resources are used for boss sounds.) Flee taunts ("Run, Coward!") are triggered at various times when the boss does certain behaviors.

Example:

```
AIDataValue ( "roarSeqs",      Resource, ( Val = ( rtIntegerArray, EXC00_ROARS      )))
AIDataValue ( "painSeqs",      Resource, ( Val = ( rtIntegerArray, EXC00_PAINS      )))
AIDataValue ( "screamSeqs",    Resource, ( Val = ( rtIntegerArray, EXC00_SCREAMS    )))
AIDataValue ( "fleeTauntSeqs", Resource, ( Val = ( rtIntegerArray, EXC00_FLEETAUNTS )))
```

4.1.1.3 rtIntegerArray Resources & Boss Sounds

```
defres rtIntegerArray ( <resource ID> )
(
   Int[] = ( Value = <resource ID> )          // An integer array
   Int[] = ( Value = <resource ID> )          // Another integer array
   Int[] = ( Value = <resource ID> )          // Have as many as you want
)
```

defres rtIntegerArray (*<resource ID>*) defines an `rtIntegerArray` resource named *<resource ID>*. The resource can have any number of integer arrays. For boss sounds, each integer array specifies the resource ID of one boss sound. Each time the boss sound behavior calls on its `rtIntegerArray` resource, a successive integer array in the resource is played (and the first array is available again once the last array is played).

The fields in the resource definition are:

- **`Int[] = ( ... )`** specifies an integer array containing the resource ID of one boss sound. The field of an `Int[]` array is:
  - **`Value = <resource ID>`** (no default) specifies the resource ID for the boss sound.

Example:

```
defres rtIntegerArray ( EXC00_ROARS )
(
   Int[] = ( Value = EXC01_SEQ_SND_ROAR1 )
   Int[] = ( Value = EXC01_SEQ_SND_ROAR2 )
   Int[] = ( Value = EXC01_SEQ_SND_ROAR3 )
)

defres rtIntegerArray ( EXC00_PAINS )
(
   Int[] = ( Value = EXC01_SEQ_SND_PAIN1 )
   Int[] = ( Value = EXC01_SEQ_SND_PAIN2 )
   Int[] = ( Value = EXC01_SEQ_SND_PAIN3 )
)

defres rtIntegerArray ( EXC00_SCREAMS )
(
   Int[] = ( Value = EXC01_SEQ_SND_SCREAM2 )
)

defres rtIntegerArray ( EXC00_FLEETAUNTS )
(
   Int[] = ( Value = EXC01_SEQ_SND_SCREAM1 )
)
```

4.1.1.4 Roar Waiting Times

```
AIDataValue ( "MinRoarTime",          Float,     ( Val = <#> ))
AIDataValue ( "MaxRoarTime",          Float,     ( Val = <#> ))
```

To define the minimum and maximum amount of time to wait to play a roar (assuming no other boss sounds have been played during that time), use these parameters:

- **`MinRoarTime`** specifies the minimum time (in seconds) the boss will wait before roaring.
- **`MaxRoarTime`** specifies the maximum time (in seconds) the boss will wait before roaring.

Example:

```
AIDataValue ( "MinRoarTime",          Float,     ( Val = 10.0 ))
AIDataValue ( "MaxRoarTime",          Float,     ( Val = 20.0 ))
```

4.1.2 ExcAlive Klown: Basic Boss Behaviors

- *Klown Code File:* excalive.fsm
- *Klown:* ExcAlive
  —*Type:* Submachine
  —*Uses External Klowns:* Difficulty, PollUpdate, MoveEngines, MoveEnginesWithLat, MoveNoSpinEngines, OrientEngines, CircleEngines, RollTowardEngines

4.1.2.1 Common Boss Sequences

```
AIDataValue ( "PlayersDeath",       Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "ExcessionsDeath",    Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "ExcessionsEntrance", Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "Collision",          Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "Damage",             Resource, ( Val = ( rtSequence, <resource ID> ))
```

The following common sequences can play on bosses::

- **PlayersDeath** (resource; no default) specifies the `rtSequence` resource that plays on the boss when the player dies.
- **ExcessionsDeath** (resource; no default) specifies the `rtSequence` resource that plays on the boss when the boss dies.
- **ExcessionsEntrance** (resource; no default) specifies the `rtSequence` resource that plays on the boss when the boss first arrives on a level.
- **Collision** (resource; no default) specifies the `rtSequence` resource that plays on the boss when the boss is in a collision.
- **Damage** (resource; no default) specifies the `rtSequence` resource that plays on the boss when the boss takes damage.

Example:

```
AIDataValue ( "PlayersDeath",       Resource, ( Val = ( rtSequence, EXC01_SEQ_VICTORY ))
AIDataValue ( "ExcessionsDeath",    Resource, ( Val = ( rtSequence, EXC01_SEQ_DEATH ))
AIDataValue ( "ExcessionsEntrance", Resource, ( Val = ( rtSequence, EXC01_SEQ_ENTER ))
AIDataValue ( "Collision",          Resource, ( Val = ( rtSequence, EXC01_SEQ_COLLISION
                ))
AIDataValue ( "Damage",             Resource, ( Val = ( rtSequence, EXC01_SEQ_DAMAGE ))
```

4.1.2.2 Boss HMO

```
AIDataValue ( "health",      Integer,   ( Val = <#> ))
AIDataValue ( "minHealth",   Float,     ( Val = <#> ))
AIDataValue ( "maxHealth",   Float,     ( Val = <#> ))
```

Bosses use `health`, `minHealth`, and `maxHealth` parameters. These work like the same named parameters for ships (see Health Klown), except that they are processed by the ExcAlive klown instead of the Health klown. The ExcAlive klown processes health and damage slightly differently than the Health klown:

- `health` has a default value of -1 instead of 1.
- Bosses do not have `spangSequence` or `deathSequence` parameters but have `Collision`, `Damage`, and `ExcessionsDeath` parameters; see Common Boss Sequences.
- There is no random factor in the damage a boss takes.
- Only Sinibomb damage affects bosses.

4.1.2.3 Maximum Distance from Home

```
AIDataValue ( "maxDistanceFromHome",   Float,   ( Val = <#> ))
```

Bosses can be told not to move beyond a set distance from their home position:

- **maxDistanceFromHome** (floating point; no default) specifies the maximum distance the boss will move from its home position.

Note: A boss's home position is specified in the klown code. It is often the center of the universe, but is set individually for each boss and can be set to waypoints, the player's position, or other positions. All of this is hardcoded.

Example:

```
AIDataValue ( "maxDistanceFromHome",   Float,   ( Val = 1500.0 ))
```

4.1.2.4 Clearing the Gate

Many bosses, upon their arrival at a jumpgate, attempt to move some distance from the gate before starting their attack. There is no AI parameter for this distance, which is hardcoded in the klowns of the bosses that clear the gate.

4.1.2.5 Attack Distances & Move to Enemy

```
AIDataValue ( "maxAttackDistance",   Float,   ( Val = <#> ))
AIDataValue ( "minAttackDistance",   Float,   ( Val = <#> ))
```

Bosses have two attack distance parameters:

- **maxAttackDistance** (floating point; no default) specifies the maximum attack distance of the boss. This distance is used by various bosses to influence their behaviors. (For example, `EXCESSION_14`'s three possible attack behaviors can be triggered only when the boss is within its `maxAttackDistance`.) There is no single overall boss behavior associated with `maxAttackDistance`.
- **minAttackDistance** (floating point; no default) specifies the minimum attack distance of the boss. When it moves toward the player (the move to enemy behavior), it tries to move to its minimum attack distance *behind* the player. In addition, `EXCESSION_14`'s combined blast wave uses `minAttackDistance`.

Note: Warriors also have `maxAttackDistance` and `minAttackDistance` parameters, but they work somewhat differently than these boss parameters.

Example:

```
AIDataValue ( "maxAttackDistance",   Float,   ( Val = 1500.0 ))
AIDataValue ( "minAttackDistance",   Float,   ( Val =  600.0 ))
```

4.1.2.6 Simple Attack

Many bosses make "simple attacks" on the player, simply firing their weapons at the player as they move or perform other behaviors.

4.1.2.7 Back Away from Enemy

```
AIDataValue ( "backAwayDistance",   Float,   ( Val = <#> ))
```

Bosses can back away from the player using:

- **backAwayDistance** (floating point; no default) specifies the distance the boss tries to move from the player when its Back Away behavior is triggered.

Example:

```
AIDataValue ( "backAwayDistance",   Float,   ( Val = 1500.0 ))
```

4.1.2.8 Lunging ("Standard Attack")

```
AIDataValue ( "lungeDistance",     Float,    ( Val = <x> ))
AIDataValue ( "lungeAngle",        Float,    ( Val = <x> ))
```

Two parameters set the Lunging behavior for a boss:

- **`lungeDistance`** (floating point; no default) specifies the distance (in meters) within which a target must be for the boss to lunge at it.
- **`lungeAngle`** (floating point; no default) specifies the cosine of the angle (1.0 = 0 degrees, 0.95 = 18 degrees, etc.; see Quick Table of Cosines) within which a target must be in front of the boss for the boss to lunge at it.

The standard attack pattern for a boss is to try to move behind the enemy at a distance equal to three times the radius of the boss. Each second, the boss checks the following:

1. If the target is within the boss's `lungeDistance` and within the `lungeAngle` in the front of the boss, the boss screams, sets its effort to 2.5, and lunges, trying to ram the player.
2. If the boss is not behind the enemy, it attempts to move behind the enemy.
3. If the target is within the `lungeDistance` but not within the `lungeAngle`, the boss tries to orient toward the target.

Lunges take priority over getting behind the enemy. For example, if the boss is not behind its target but the target is within the boss's `lungeDistance` and `lungeAngle`, the boss lunges.

(Note that the lunge distance is also used for the Clumsy Charge behavior, so if a boss has both Clumsy Charge and Lunge behaviors, it must use the same lunge distance parameter. Few bosses, however, have both behaviors.)

Example:

```
AIDataValue ( "lungeDistance",     Float,    ( Val = 1500.0 ))
AIDataValue ( "lungeAngle",        Float,    ( Val =    0.95 ))
```

4.1.2.9 Clumsy Charge

```
AIDataValue ( "lungeDistance",       Float,    ( Val = <#> ))
AIDataValue ( "peelOffAngle",        Float,    ( Val = <#> ))
AIDataValue ( "chargeOvershoot",     Float,    ( Val = <#> ))
```

These parameters allow bosses to make clumsy charges at the player:

- **`lungeDistance`** (floating point; no default) specifies the distance in meters the boss can lunge. When a boss charges, the first thing it does is checks to see if its target lies within its lunge distance:
  If the target is further away, the boss turns and moves directly toward the target.
  If the target initially lies within the lunge distance, the boss orients (without lateral movement) until it faces the target. The boss then chooses a fixed direction (see `peelOffAngle` below) and moves along that direction instead of moving directly at the enemy.
- **`peelOffAngle`** (floating point; default is 0.0) specifies direction the boss moves when it charges. This is an angle in degrees from the line from the boss to the charge target. If the peel-off angle is 0 (the default), the

direction is directly toward and through the target. Otherwise, the peel-off angle describes the angle of an upward pitch from the boss's current orientation, which is be used as the target direction.

- **chargeOvershoot** (floating point; no default) specifies the distance in meters from the targets initial position (its position at the start of the boss's charge) at which a charging boss ends its charge.

(Note that the lunge distance is also used for the Lunging ("Standard Attack") behavior, so if a boss has both Clumsy Charge and Lunge behaviors, it must use the same lunge distance parameter. Few bosses, however, have both behaviors.)

Example:

```
AIDataValue ( "chargeOvershoot",     Float,    ( Val = 800.0 ))
AIDataValue ( "lungeDistance",       Float,    ( Val = 500.0 ))
AIDataValue ( "peelOffAngle",        Float,    ( Val =  30.0 ))
```

4.1.2.10 Fleeing

Note: There is also a separate running away common boss behavior, see Running Away.

```
AIDataValue ( "fleeDistance",           Float,    ( Val = <#> ))
```

Many bosses flee from time to time. Use this parameter to set fleeing distance:

- **fleeDistance** (floating point; no default) specifies the distance (in meters) the boss tries to move from the player. (When a boss flees, it picks a random point on a sphere of a specified radius, the flee distance, away from the container it's fleeing from. It also makes sure that the random point lies in the hemisphere on the "near" side of the container. For example, if the boss is at Y = 1000, and the container it's running from is at Y = 0, it'll pick a point on the sphere that has Y > 0.)

Note: Warriors also have a `fleeDistance` parameter, but it different than the boss parameter.

Example:

```
AIDataValue ( "fleeDistance",           Float,    ( Val = 1500.0 ))
```

4.1.2.11 Running Away

Note: The running away behavior re-uses the `fleeDistance` parameter of the Fleeing common boss behavior. It uses this parameter to set the distance to which it runs away from the target object that triggers the behavior. The rest of the running away behavior depends upon whether or not the boss is already moving away from the target.

4.1.2.11.a Not Moving Away from the Target

```
AIDataValue ( "runAwayMinAngle",          Float,    ( Val = <#> ))
AIDataValue ( "runAwayMaxAngle",          Float,    ( Val = <#> ))
```

When the boss is not already moving away from the target, `runAwayMinAngle` and `runAwayMaxAngle` specify two cones (which are defined relative to the target object). The boss chooses a point that is `fleeDistance` away from the target and is outside `runAwayMinAngle` but inside `runAwayMaxAngle`. The boss then runs away to this point. (The boss chooses to go to the right or the left side of the target if it is already located on the that side.) The parameters are:

- **runAwayMinAngle** (floating point; default is pi / 3.0) specifies the angle (in radians; a radian is approximately 57°17') that forms the minimum run-away cone from its target. The boss will not choose a run away point inside this cone.

- **runAwayMaxAngle** (floating point; default is 2 * pi / 3.0) specifies the angle (in radians) that forms the maximum run-away cone from its target. The boss will choose a run away point that is inside this cone but outside the minimum run-away cone.

Example:

```
AIDataValue ( "runAwayMinAngle",           Float,    ( Val = 1.043 ))
AIDataValue ( "runAwayMaxAngle",           Float,    ( Val = 2.094 ))
```

4.1.2.11.b Moving Away from the Target

```
AIDataValue ( "runAwayRelVelDevAngle",     Float,    ( Val = <#> ))
```

If the boss is already moving away from the target when the running away behavior is triggered, it continues to do so, going to a point that is `fleeDistance` away from the target. The code checks how well the boss is moving away when the behavior is triggered and adjusts this if the boss can run away better than what it is currently doing. This adjustment includes a parameter that allows a random deviation in the point selected as the run-away point:

- **runAwayRelVelDevAngle** (floating point; default is pi / 10.0) specifies the maximum random angle (in radians) that is used when selecting the boss's run-away point. (The actual random angle will vary between 0.0 and `runAwayRelVelDevAngle`.)

Example:

```
AIDataValue ( "runAwayRelVelDevAngle",     Float,    ( Val = 0.314 ))
```

4.1.2.12 Park at Gate

Bosses can use the Park at Gate behavior to move near a jumpgate and remain there. For example, some bosses use this behavior while the player is cloaked. When using this behavior, a boss moves towards the closest jumpgate until it is within three of the boss's radii of gate and then remain there until another behavior is triggered.

4.1.2.13 Roid Inspector

```
AIDataValue ( "RIRoidSensor",       Resource, ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "minRIDistance",      Float,    ( Val = <#> ))
AIDataValue ( "maxRIWorldDistance", Float,    ( Val = <#> ))
AIDataValue ( "minRIMass",          Float,    ( Val = <#> ))
```

Bosses have an idle behavior they can use when the player is cloaked. The behavior picks random roids for the boss to inspect and drives the boss from one roid to another. One interesting side effect to consider when using this behavior is that a boss with a Lightning Gun, for example, blows up roids all over the place! The following parameters are used to choose roids and assign them as target locations for the boss to go inspect:

- **RIRoidSensor** (resource; default is no sensor) specifies the type of detector a boss uses to detect asteroids for its roid inspector behavior. See Sensors for more details on sensors.
- **minRIDistance** (floating point; default is 1300.0) specifies the minimum distance (in meters) an asteroid must be from the boss's current location. Asteroids closer than this distance will not be selected for inspection.
- **maxRIDistance** (floating point; default is 3000.0) specifies the maximum distance (in meters) an asteroid must be from the boss's current location. Asteroids farther than this distance will not be selected for inspection.

- **minRIMass** (floating point; default is 1000.0) specifies the minimum mass (in kilograms) the asteroid. Asteroids with masses below this value will not be selected for inspection.

Note: Please modify these parameters with extreme care and cautiousness!

Example:

```
AIDataValue ( "RIRoidSensor",       Resource, ( Val = ( rtBodySensor, EXCRI_ROID_SENSOR
                ) ))
AIDataValue ( "minRIDistance",      Float,    ( Val = 1300.0 ))
AIDataValue ( "maxRIWorldDistance", Float,    ( Val = 3000.0 ))
AIDataValue ( "minRIMass",          Float,    ( Val = 1000.0 ))
```

4.1.3 Circling the Player or a Location

Use the CircleEngines or CircleEnginesNoSpin klown to allow a boss to circle the player or a location. (The ExcAlive klown already uses the CircleEngines klown.) The engines allow the circling motion around a position (set by `circleDistance`), while the boss's klown specifies whether this position is the player or a location in space.

4.1.4 SelectiveDamage Klown

- *Klown Code File:* selectivedamage.fsm
- *Klown:* SelectiveDamage
  —*Type:* Submachine

```
AIDataValue ( "jointSelectiveDamage", Resource, ( Val = ( rtJointSelectiveDamage,
                <resource ID> ) ))
AIDataValue ( "damageColor",          Integer,  ( Val = ( <RGBA macro> ))
AIDataValue ( "damageTime",           Float,    ( Val = ( <#> ))
```

Bosses can have selective damage, which allows joints of the boss to receive and track damage (from Sinibombs) separately from other joints. For bosses with selective damage, a single parameter must be specified, and two further parameters can be specified:

- **jointSelectiveDamage** specifies an `rtJointSelectiveDamage` resource. Only collisions against bounding boxes attached to the joints listed in the `rtJointSelectiveDamage` resource cause damage to the joints. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtJointSelectiveDamage` resources.
- **damageColor** (RGBA macro; default is RGBA(255, 0, 0, 255)) specifies the RGBA color (and can include alpha) that flashes on the joint when the joint takes damage.
- **damageTime** (floating point; default is 2.0) specifies the time (in seconds) the `damageColor` plays.

Note: `damageColor` and `damageTime` are also used in the SkeletonSelectiveDestruction klown. Note, however, that `damageTime` has a different default value in that klown.

Example:

```
AIDataValue ( "jointSelectiveDamage", Resource, ( Val = ( rtJointSelectiveDamage,
                EXCESSION_05 ) ))
```

Bosses which use selective damage should also have target points specified at which Sinibombs target. The way to do this is to add dummy objects in the boss's .MAX file labeled with "`attribute=target`". The Sinibombs guide toward the closest target attribute on the boss.

4.1.5 Anti-Missile Detectors & Defenses

Many bosses have anti-missile detectors, which detect missiles and Sinibombs (and can be set to detect only Sinibombs). When the detectors sense a valid target, a variety of different behaviors can be triggered, from firing on the missile, fleeing the missile, using a tractor beam on the missile, and so on per the bosses' specific behaviors.

4.1.5.1 AntiMissileDectector Klown

- *Klown Code File:* specialweapon.fsm
- *Klown:* AntiMissileDetector
  —*Type:* Submachine

```
AIDataValue ( "onlyDetectSinibombs", Integer,  ( Val = <#> ))
```

Bosses that have anti-missile defenses use the AntiMissileDetector klown. The klown can be set so that only Sinibombs are detected (rather than any missile at all):

- **`onlyDetectSinibombs`** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) only Sinibombs are detected.

Note: Typically, you'll want bosses only to detect Sinibombs. If you bosses to detect all types of missiles, then omit having an `onlyDetectSinibombs` parameter, as its default setting has the sensor detect all missiles.

Example:

```
AIDataValue ( "onlyDetectSinibombs", Integer,  ( Val = 1 ))
```

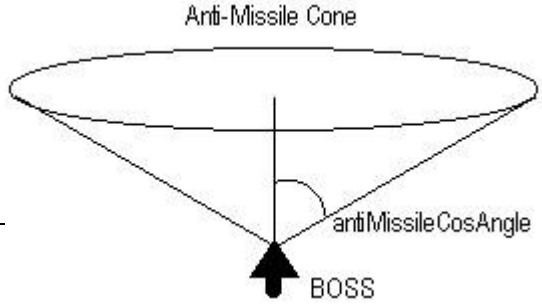
4.1.5.2 AntiMissileDefense Klown

- *Klown Code File:* antimissile.fsm
- *Klown:* AntiMissileDefense
  —*Type:* Submachine
  —*Uses External Klowns:* PollFast, Team

```
AIDataValue ( "antiMissileSensor",   Resource, ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "antiMissileCosAngle", Float,    ( Val = <#> ))
AIDataValue ( "antiMissileCategory", Integer,  ( Val = <resource ID> ))
```

The AntiMissileDefense klown can detect missiles (or just Sinibombs) and trigger the boss's anti-missile defense behavior. If the `antiMissileCategory` parameter is specified, then the boss has an anti-missile weapon that it fires at the missile. Otherwise, the boss will adopt another behavior (per its klown; typically, fleeing behavior is triggered). The parameters are:

- **`antiMissileSensor`** (resource; default is no sensor) specifies the type of anti-missile detector the boss uses. See Sensors for more details on sensors.
- **`antiMissileCosAngle`** (floating point; default is 0.0) specifies the cosine (see Quick Table of Cosines) of



the angle that forms the antimissile cone in which the sensor searches for missiles.

- **antiMissileCategory** (resource ID; default is -1, no category) specifies the resource ID of the weapon category for the boss's anti-missile weapon. (To fire on missiles, the boss must have an anti-missile weapon attached, and this resource ID indicates its weapon category.)

Example:

```
AIDataValue ( "antiMissileSensor",   Resource, ( Val = ( rtBodySensor,
                  EXC01_ANTIMISSILE)))
AIDataValue ( "antiMissileCosAngle", Float,    ( Val = 0.25 ))
AIDataValue ( "antiMissileCategory", Integer,  ( Val = GUNCAT_ANTIMISSILE ))

defres rtBodySensor ( EXC01_ANTIMISSILE )
(
   Type.Simple =
   (
      MaxDistance = 1000.0
      SenseType   = kSenseMissile
   )
)
```

4.1.5.3 TractorAntiMissile Klown

- *Klown Code File:* antimissile.fsm
- *Klown:* TractorAntiMissile
  —*Type:* Submachine
  —*Uses External Klowns:* TractorBeam2, PollFast, Team

```
AIDataValue ( "antiMissileSensor",   Resource, ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "antiMissileCosAngle", Float,    ( Val = <#> ))
```

The TractorAntiMissile klown can detect missiles (or just Sinibombs) and trigger the boss's tractor beam to attempt to repel the incoming object. This klown uses two parameters that also appear in the AntiMissileDefense klown:

- **antiMissileSensor** (resource; default is no sensor) specifies the type of anti-missile detector the boss uses. See Sensors for more details on sensors.
- **antiMissileCosAngle** (floating point; default is 0.0) specifies the cosine of the angle (see Quick Table of Cosines) that forms the antimissile cone in which the sensor searches for missiles.

Example:

```
AIDataValue ( "antiMissileSensor", Resource, ( Val = ( rtBodySensor, EXC07_ANTI_MISSILE
                  )))
AIDataValue ( "antiMissileCosAngle", Float,  ( Val = 0.25 ))
```

4.1.6 ChargingMainWeapon Klown

- *Klown Code File:* specialweapon.fsm
- *Klown:* ChargingMainWeapon
  —*Type:* Submachine

```
AIDataValue ( "mainWeaponCharge",     Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "mainWeaponChargeTime", Float,    ( Val = <#> ))
```

Some bosses have a main weapon which must be charged up before they can be fired. These parameters control main charging weapons:

- **mainWeaponCharge** (resource; no default) specifies an `rtSequence` resource that plays while the weapon is charging. The sequence may also emit a single object that exists until the weapon is actually fired, whereupon this object is destroyed. (This object is usually some focussing effect that should be removed once the shot is fired.) For this to happen, the sequence must contain an `rtEmitInfo` action named "`effectEmitter`" as shown in the example below. (This name is indeed a name for the action and allows klowns to find the action, and it is not a resource ID.) See the separate documents: The Sequencing System on how to define sequences (and name actions) and Emitters on how to define `rtEmitInfo` resources.
- **mainWeaponChargeTime** (floating point; default is 1.0) specifies the amount of time (in seconds) the main weapon must charge before it can be fired.

Example:

```
AIDataValue ( "mainWeaponCharge",     Resource, ( Val = ( rtSequence,
                  EXC19_MAIN_WEAPON_SEQ )))
AIDataValue ( "mainWeaponChargeTime", Float,    ( Val = 5.5 ))

defres rtSequence ( EXC19_MAIN_WEAPON_SEQ )
(
   Actions [] =
   (
      ActionId = ( rtEmitInfo,    EXC19_MAIN_WEAPON_ACT1 )
      Time     = 0
      Name     = "effectEmitter"
   )
)

defres rtEmitInfo ( EXC19_MAIN_WEAPON_ACT1 )
(
   Emitter.EmitSingle = ( ContainerId = EXC19_CHARGE_EFFECT )
   INIT_POS_POINT_ORIGIN()
   INIT_ORIENT_FIXED_IDENTITY()
   INIT_VEL_FIXED( 0.0, 0.0, 0.0 )
   INIT_COLOR( 255, 255, 255, 0 )
   INIT_RIGID_HIERARCHY_PARENT()
)
```

4.1.7 BlastWave Klown

- *Klown Code File:* specialweapon.fsm
- *Klown:* BlastWave
  —*Type:* Submachine
- *Note:* Some of the parameters appear directly in weapon.cpp and have their defaults set there, rather than through a klown.

```
AIDataValue ( "blastDamage",         Float,    ( Val = <#> ))
AIDataValue ( "MaxForce",            Float,    ( Val = <#> ))
AIDataValue ( "BlastWidth",          Float,    ( Val = <#> ))
AIDataValue ( "BlastWaveTime",       Float,    ( Val = <#> ))
AIDataValue ( "blastSequence",       Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "MaxBlastRandomForce", Float,    ( Val = <#> ))
```

```
AIDataValue ( "blastSensor",          Resource, ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "MaxBlastTorque",       Float,    ( Val = <#> ))
AIDataValue ( "MaxBlastShake",        Float,    ( Val = <#> ))
AIDataValue ( "blastChargeTime",      Float,    ( Val = <#> ))
```

Blast waves are used by certain bosses and by concussion shots. A blast wave can be thought of as a ring which expands out from the detonation point. As the ring hits objects, it damages them and pushes them away from the epicenter of the blast. A blast wave creates a visual effect while applying damage and outward force to objects caught within the wave. The strength of the force and the amount of damage are greatest at the epicenter of the blast wave and drop off exponentially with distance from the epicenter:

$$e^{-r}$$

where r is the distance from the epicenter.

The parameters for blast waves are:

- **blastDamage** (floating point; default is 0.0) is the amount of damage applied to objects at the epicenter. Damage to an object is applied just once: when the wave first hits the object. (Even though ***forces*** will still be applied as the wave passes through the object, ***damage*** will not be continually applied.) As explained above, the damage amount decreases exponentially with distance from the epicenter of the blast. (The Sinistar are immune from blast damage.)
- **MaxForce** (floating point; default is 1.0) is the maximum outward force at the epicenter of the blast. The force, like damage, decreases exponentially with distance from the epicenter of the blast.
- **BlastWidth** (floating point; default is 1.0) measures the width of the blast wave. Forces are applied to all objects within the blast wave width.
- **BlastWaveTime** (floating point; default is 1.0) measures how long it takes the wave to get from the epicenter to the furthest point it can reach.
- **blastSequence** (resource; no default) specifies the `rtSequence` resource used to generate the visual effect of the blast wave.
- **MaxBlastRandomForce** (floating point; default is 0.0) specifies the maximum force that randomly fluctuates in the outward force. At any moment while in a blast wave, the force applied to an object is thus the `MaxForce` as adjusted for distance plus a randomly chosen value equal to or greater than 0.0 and less than or equal to `MaxBlastRandomForce`. The higher the number, the more the ship gets buffeted around.
- **blastSensor** (resource; default is no sensor) specifies the sensor that is used in two ways for blast waves: 1) It specifies the types of objects that are affected by the blast wave. 2) The maximum range of the sensor is also the furthest distance the blast wave will reach. See Sensors for more details on sensors.
- **MaxBlastTorque** (floating point; default is 0.0) is a measure of how much random torque gets applied to the ship and also produces a buffeting effect.
- **MaxBlastShake** (floating point; default is 0.0) specifies how much the camera shakes as the blast wave travels through it.
- **blastChargeTime** (floating point; default is 0.0) specifies the amount of time (in seconds) the blast weapon requires for recharging. Once fired, the weapon cannot fire again until its `blastChargeTime` has passed.

Example:

```
AIDataValue ( "blastDamage",          Float,    ( Val = 18 ))
AIDataValue ( "MaxForce",             Float,    ( Val = 1000000 ))
AIDataValue ( "BlastWidth",           Float,    ( Val = 400 ))
AIDataValue ( "BlastWaveTime",        Float,    ( Val = 1.5 ))
AIDataValue ( "blastSequence",        Resource, ( Val = ( rtSequence, EXC18_BLAST_SEQ )))
```

```
AIDataValue ( "MaxBlastRandomForce", Float,    ( Val = 90000 ))
AIDataValue ( "blastSensor",         Resource, ( Val = ( rtBodySensor, EXC18_BLAST_SENSOR
                ) ))
AIDataValue ( "MaxBlastTorque",      Float,    ( Val = 900000 ))
AIDataValue ( "MaxBlastShake",       Float,    ( Val = 0.1 ))
AIDataValue ( "blastChargeTime",     Float,    ( Val = 8.0 ))
```

4.1.8 MineControl Klown: Mine Laying

- *Klown Code File:* minecontrol.fsm
- *Klown:* MineControl
  —*Type:* Submachine
  —*Uses External Klowns:* Weapons

```
AIDataValue ( "maxMines",     Integer,  ( Val = <#> ))
AIDataValue ( "minMines",     Integer,  ( Val = <#> ))
AIDataValue ( "mineCategory", Integer,  ( Val = <resource ID> ))
```

Some bosses lay seeking mines, which when activated home in on the player. The following parameters control mine laying:

- **maxMines** (integer; no default) specifies the maximum total number of mines laid by a boss that may be in play. (Only mines "in play" count—a mine that is destroyed after being laid no longer counts.) The boss lays mines until the `maxMines` limit is reached, whereupon it stops laying mines (but see the next parameter below).
- **minMines** (integer; no default) specifies the minimum total number of mines that must be in play for the boss to resume mine laying once it has stopped laying mines. For example, `maxMines` with `Val = 100` and `minMines` with `Val = 90` mean that the boss lays mines until 100 are in play. It then stops laying mines until there are only 90 in play, whereupon it resumes laying mines until 100 are again in play.
- **mineCategory** (resource ID; no default) specifies the resource ID of the weapon category for the boss's mine weapon. (To lay mines, the boss must have a mine weapon attached, and this resource ID indicates its weapon category.)

Note: Both `EXCESSION_10` and `EXCESSION_10B` (see Launching Warriors in each boss's section) use these parameters to control the number of warriors (not mines) they launch. `EXCESSION_10B` (see Mine Laying) also uses these parameters to control mine laying as described above.

Example:

```
AIDataValue ( "maxMines",     Integer,  ( Val = 100 ))
AIDataValue ( "minMines",     Integer,  ( Val =  90 ))
AIDataValue ( "mineCategory", Integer,  ( Val = GUNCAT_MINES ))
```

4.1.9 Boss Health & Gate Damage

- *Klown Code File:* excegg.fsm
- *Klown:* ExcEgg
  —*Type:* HFSM
  —*Uses External Klowns:* various, see Jumpgates

```
AIDataValue ( "BossHealthChart", Resource, ( Val = ( rtIntegerArray, <resource ID> )))
```

The ExcEgg klown controls the jumpgates, and damage on a gate can affect the level's boss. Before a boss arrives on a level, its health decreases though a timer mechanism before its arrival. Thus, the longer the player delays the arrival of the boss by damaging the gate, the lower the boss's health can drop.

The computation of the health is based on interpolating time keys with percentage of health. The boss has a set of time keys, with the boss's health percentage specified for each time key. For any specific time, the boss's current health percentage is interpolated (and displayed on the boss health meter).

The gate AI in egg.r points to the resource that specifies the time keys and health percentages:

- **BossHealthChart** (resource; no default) specifies the `rtIntegerArray` resource that contains the time keys and health percentages.

Example:

```
AIDataValue ( "BossHealthChart", Resource, ( Val = ( rtIntegerArray, GATE_29 )))
```

The `rtIntegerArray` resource specified by `BossHealthChart` contains the time key and health percentages. This resource must be organized in pairs of time keys/health percentages. For each pair, the time key must be listed first and its health percentage second.

Example:

```
defres rtIntegerArray ( GATE_29 )
(
   Int[] = ( Value = 0 )       // Time key; time in sec
   Int[] = ( Value = 100 )     // Health percentage at time key

   Int[] = ( Value = 120 )     // Time key
   Int[] = ( Value = 91 )      // Health percentage at time key

   Int[] = ( Value = 360 )     // Time key
   Int[] = ( Value = 50 )      // Health percentage at time key

   Int[] = ( Value = 720 )     // Time key
   Int[] = ( Value = 25 )      // Health percentage at time key
)
```

When the level time is between 0 and 120 seconds, the boss's health is proportionally computed between 100 and 91 percent. When the current level time is between 120 and 360 seconds, the boss's health is proportionally computed between 91 and 50. And so on. For example, if the level time is 240, then the boss health is set to 70.5 percent health.

4.1.10 EXCESSION_00 Dummy Boss

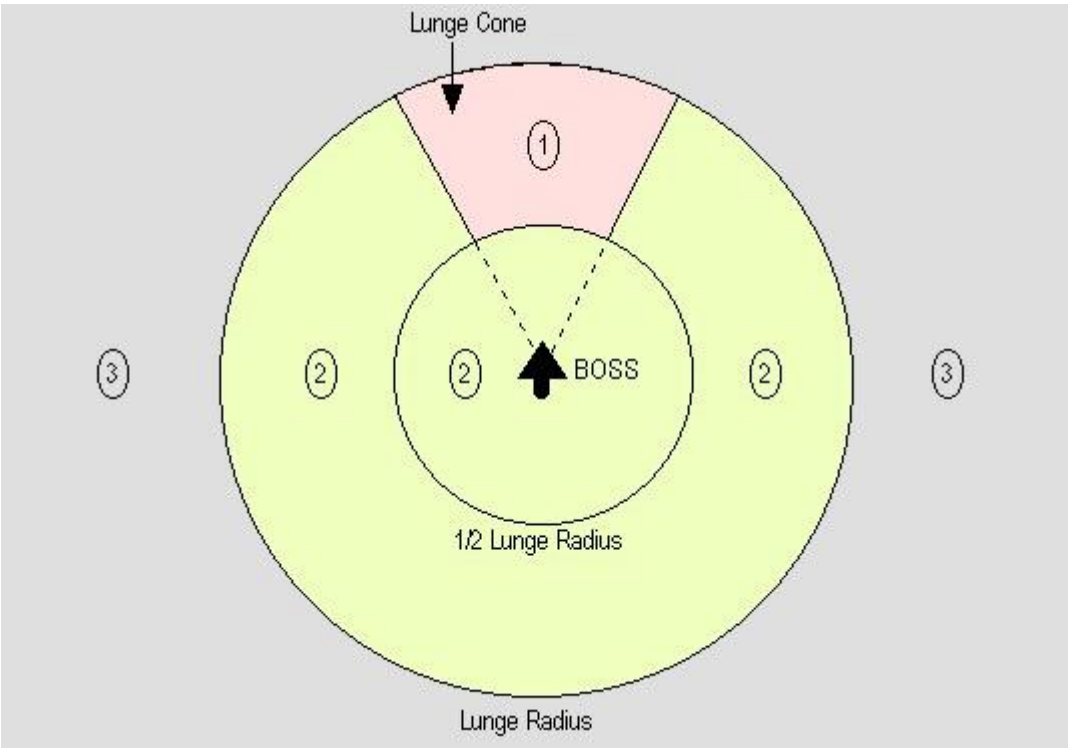
- *Klown Code File:* exc00.fsm
- *Klown:* DEvil_00
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SetTarget, FireControl, Weapons, SearchForTarget, AntiMissileDefense, TargetOffset, ExcAlive, ExcSound, Scorable
  —*Has Internal Klown:* NormalBehavior

DEvil_00 is a klown that was used to dummy up bosses before the bosses' specific klowns had been developed.

The behaviors for this klown are:

- Moves toward the player if the player is too far away.
- Can lunge at the player.
- Can fire on the player.

4.1.10.1 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is in the lunge cone, within the lunge radius, and further than one-half the lunge radius of the boss , the boss lunges at the player (100% chance).
2	Yellow Zone: If the player is within the lunge radius but outside the red zone, the boss aims and fires its weapon on the player (100% chance).
3	Gray Zone: If the player is outside the lunge radius, the boss moves toward the player:

4.1.10.2 Lunging

The boss can lunge at the player using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2 Boss-Specific Behaviors

Bosses are listed by their container resource IDs. Parenthetical information after the boss ID notes the level the boss appears on (although this will change for some bosses) and their old, less meaningful boss numbers from earlier design documentation.

Bosses will be shifted among levels based on how hard the various bosses turn out to be. This document reflects their May 1999 assignments, with `EXCESSION_10B` on Level 22, plus known intended final assignments. (Boss are being temporarily shifted around during tuning.)

ZONE	GAME LEVEL	BOSS	EXCESSION_#	Boss #	ASSIGNED
WORKER	1	Porky	01	1	Rafael
	2	Angler	02	2	Rafael
	3	Zapp	09	7	Sebastien
	4	(no boss)	-	-	
WARRIOR	5	Spike	03	3	Sebastien
	6	Grinder	07	6	Rafael
	7	Scarab	10	8	Sebastien
	8	(no boss)	-	-	
PLANETOID	9	Scarab II	10B	8b	Sebastien
	10	Fan	05	4	Brian
	11	Single Twin	19	15	Brian
	12	(no boss)	-	-	
VOID	13	Larva (Space Hen)	06	5	Brian
	14	Mine Layer	14	11	Sebastien
	15	Lightbulb	17	13	Sebastien
	16	(no boss)	-	-	
GRAVEYARD	17	Brain	18	14	Brian
	18	Bull	11	9	Sebastien
	19	Cluster	15	12	Brian
	20	(no boss)	-	-	
SINISTAR	21	Mother Ship	13	10	Brian
	22	Phoenix	23	16a	Brian
	23	Double Twin	19	15	Brian
	24	The Sinistar	24	17	Brian

4.2.1 EXCESSION_01 “Porky”

- *Klown Code File:* exc01.fsm
- *Klown:* DEvil_01
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SphericalRestriction, SetTarget, FireControl, Weapons, SearchForTarget, AntiMissileDefense, ExcAlive, ExcSound, Scorable, Exc01_BouncingSpikes, ExcPain
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has articulated spikes and whiskers.
- Has an anti-missile weapon that can fire at Sinibombs.
- Can lunge at the player.

4.2.1.1 Articulation of Whiskers & Spikes

- *Klown Code File:* SpringyThings.fsm
- *Klown:* Exc01_BouncingSpikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton, ExcPain

```
AIDataValue ( "whiskerResistance",   Float,    ( Val = <#> ))
AIDataValue ( "spikeResistance",     Float,    ( Val = <#> ))
AIDataValue ( "spikeNoise",          Float,    ( Val = <#> ))
AIDataValue ( "spikeNoiseInterval",  Float,    ( Val = <#> ))
```

Porky may be a slow, stupid pig, but boy can he wiggle his whiskers and spikes in a way that’d fill a warthog with envy! Porky uses several of the Common Articulation Parameters (`jointDamping`, `painReaction`, `painReactionTime`). In addition, he uses the following:

- **`whiskerResistance`** (floating point; default is 300.0) specifies the stiffness of the whiskers (higher values increase stiffness).
- **`spikeResistance`** (floating point; default is 100.0) specifies the stiffness of the spikes (higher values increase stiffness).
- **`spikeNoise`** (floating point; default is 0.3) specifies how much noise (randomness) to add to the wiggling of the spikes when the boss is at rest. Low values mean the spikes wriggle little or not all; high values mean they thrash about.
- **`spikeNoiseInterval`** (floating point; default is 0.5) specifies how often (the interval in seconds) to add noise to the wiggling of the spikes when the boss is at rest.

Use `whiskerResistance`, `spikeResistance`, and the common `jointDamping` to control the whiskers and spikes when the boss is moving. Use `spikeNoise` and `spikeNoiseInterval` to control the spikes when the boss is at rest.

Example:

```
AIDataValue ( "whiskerResistance",   Float,    ( Val = 150.0 ))
AIDataValue ( "spikeResistance",     Float,    ( Val =  60.0 ))
AIDataValue ( "jointDamping",        Float,    ( Val =   0.8 ))
```

```
AIDataValue ( "spikeNoise",          Float,    ( Val =   0.25 ))
AIDataValue ( "spikeNoiseInterval",  Float,    ( Val =   0.3 ))
AIDataValue ( "painReaction",        Float,    ( Val =   2.0 ))
AIDataValue ( "painReactionTime",    Float,    ( Val =   2.0 ))
```

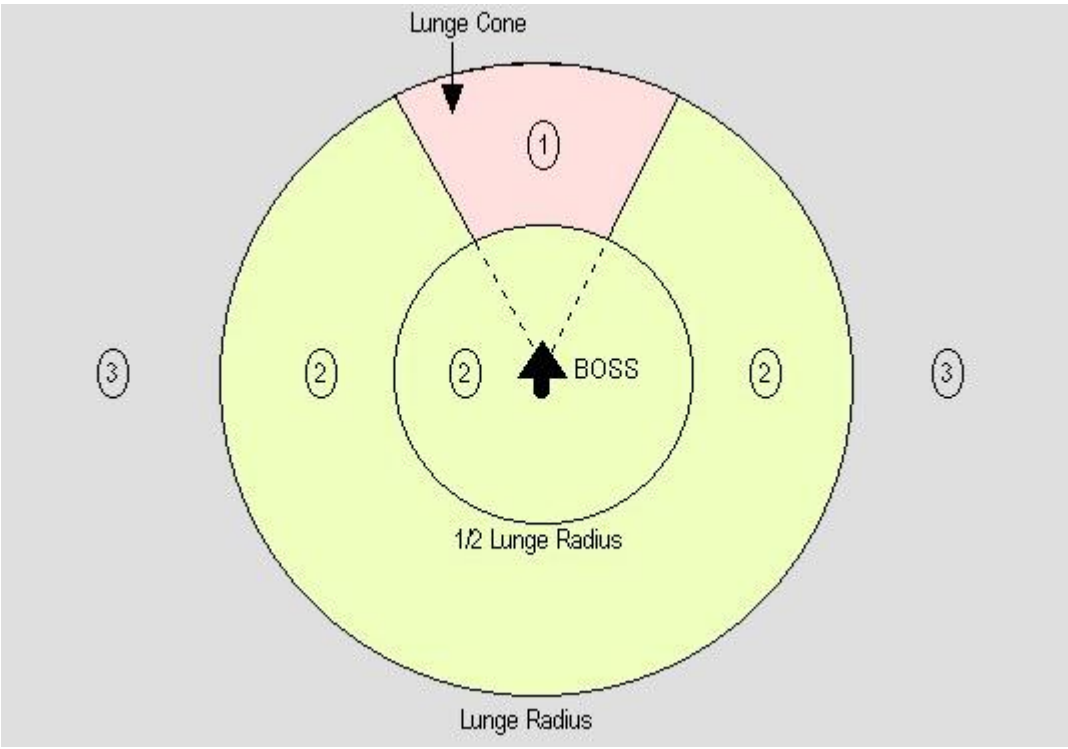
4.2.1.2 Weaponry

The boss currently has a green plasma missile as its default weapon (although a rolling ball weapon is commented out its .r file) and an anti-missile weapon that is uses against Sinibombs.

4.2.1.3 Anti-Sinibomb Defenses

This boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCategory`, `onlyDetectSinibombs`) to detect incoming Sinibombs and fire its anti-missile weapon on them.

4.2.1.4 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is in the lunge cone, within the lunge radius, and further than one-half the lunge radius of the boss , the boss lunges at the player (100% chance).
2	Yellow Zone: If the player is within the lunge radius but outside the red zone, the boss aims and fires its weapon on the player (100% chance).
3	Gray Zone: If the player is outside the lunge radius, the boss moves toward the player:

4.2.1.5 Lunging

The boss can lunge at the player using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2.2 EXCESSION_02 “Angler”

- *Klown Code File:* exc02.fsm
- *Klown:* DEvil_02
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SphericalRestriction, SetTarget, FireControlArray, Weapons, SearchForTarget, AntiMissileDefense, TargetOffset, ExcAlive, ExcSound, Scorable, StrikingArms, Exc02_WavingSpikes, ExcPain
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has an articulated head and striking arm.
- Has an anti-missile weapon that can fire at Sinibombs.
- Runs away from the player when it takes damage from Sinibombs.
- Can angrily fire on the player if it takes damage while running away.
- Otherwise it tries to approach and fire on the player.

4.2.2.1 Arm & Head Articulations

4.2.2.1.a Pain Reaction

The boss uses the Common Articulation Parameters (`painReaction`, `painReactionTime`).

4.2.2.1.b Head Sweeping Parameters

The boss’s head uses the common Head Sweeping parameters (`headMaxAngle`, `headSweepAngle`, `headSweepSpeed`, `headTargetOffset`).

4.2.2.1.c Striking Arm Parameters

- *Klown Code File:* wavingSpikes.fsm
- *Klown:* Exc02_Waving Spikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
AIDataValue ( "armResistance",        Float,    ( Val = <#> ))
AIDataValue ( "armDamping",           Float,    ( Val = <#> ))
```

The boss’s arm uses the Striking Arms parameters (`armMaxAngle`, and all the other arm parameters). In addition, the Exc02_WavingSpikes klown in wavingSpikes.fsm also contains two arm parameters, which control the resistance and damping of the arms, separate from the common joint resistance and damping parameters:

- **`armResistance`** (floating point; default is 50.0) specifies the stiffness of the arm’s movement.
- **`armDamping`** (floating point; >= 0.0 to <= 1.0; default is 0.4) specifies the arm’s damping factor, which is the time in seconds the articulation movement takes to come to rest.

Example:

```
AIDataValue ( "armResistance",      Float, ( Val = 50.0 ))
AIDataValue ( "armDamping",         Float, ( Val =  0.4 )))
```

4.2.2.1.d Spikes

- *Klown Code File:* wavingSpikes.fsm
- *Klown:* Exc02_Waving Spikes
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
AIDataValue ( "spikeBaseResistance",        Float,    ( Val = <#> ))
```

The boss's arm uses the Striking Arms parameters (`armMaxAngle`, and all the other arm parameters). In addition, the Exc02_WavingSpikes klown in wavingSpikes.fsm also contains two arm parameters, which control the resistance and damping of the arms, separate from the common joint resistance and damping parameters:

- **`armResistance`** (floating point; default is 50.0) specifies the stiffness of the arm's movement.
- **`armDamping`** (floating point; >= 0.0 to <= 1.0; default is 0.4) specifies the arm's damping factor, which is the time in seconds the articulation movement takes to come to rest.

Example:

```
AIDataValue ( "spikeBaseResistance",     Float, ( Val = 800.0 ))
```

4.2.2.2 Weaponry

Angler has a beam weapon and long-range missiles, both of which it fires at the player. The boss also has anti-missile missiles, which it fires at Sinibombs.

4.2.2.3 Anti-Sinibomb Defenses

This boss detects incoming Sinibombs with the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCategory`, `antiMissileCosAngle`) and fires its anti-missile weapon on them.

4.2.2.4 Behavior Summary

Approaching	When the player is outside the boss's `minAttackDistance`, the boss moves toward the player, trying to get behind the player and within the `minAttackDistance`.
Firing	When the player is within the `minAttackDistance`, the boss fires on the player.
Running Away	If the boss takes damage from a Sinibomb, the boss runs away (see Brave Sir Angler Ran Away).
Angry Firing	If the boss is running away and takes damage from a Sinibomb, it stops running away and angrily fires on the player (see Angry Firing).

4.2.2.5 Brave Sir Angler Ran Away

```
AIDataValue ( "retreatDistance",   Float,    ( Val = <#> ))
```

When the boss takes damage from a Sinibomb, it immediately runs away:

- **`retreatDistance`** (floating point; default is 2500.0) is the distance the boss retreats from the player's position (the player's position at the instant the boss takes damage from a Sinibomb). (The boss chooses a point `retreatDistance` away from the player, moves there, and then resumes its normal attack behavior.)

The boss has its effort level set to 3.5 while running away.

Example:

```
AIDataValue ( "retreatDistance",   Float,   ( Val = 2500.0 ))
```

4.2.2.6 Angry Firing

The boss has its effort level set to 1.5 while it is in its angry firing state. This is not tunable.

4.2.3 EXCESSION_03 “Spike”

- *Klown Code File:* exc03.fsm
- *Klown:* DEvil_03
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, Weapons, AntiMissileDefense, ExcAlive, ExcSound, Attached, Skeleton, StartWithShield
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has a selective damage point on its abdomen.
- Has articulated spikes.
- Has shields. They fade out when it wants to fire, and fade right back in again.
- Upon arrival in the level, moves 800 meters from the gate.
- Has a coordinated spike attack in which it fires photon guns out of its spikes.
- It can chase the player for a time.
- If the player gets too close, it tries to get some distance away, turns around, and fires all its guts.
- It is able to lunge at the player.

4.2.3.1 Selective Damage

The boss has a selective damage point on its abdomen, which is its only defense from Sinibombs. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.3.2 Shields

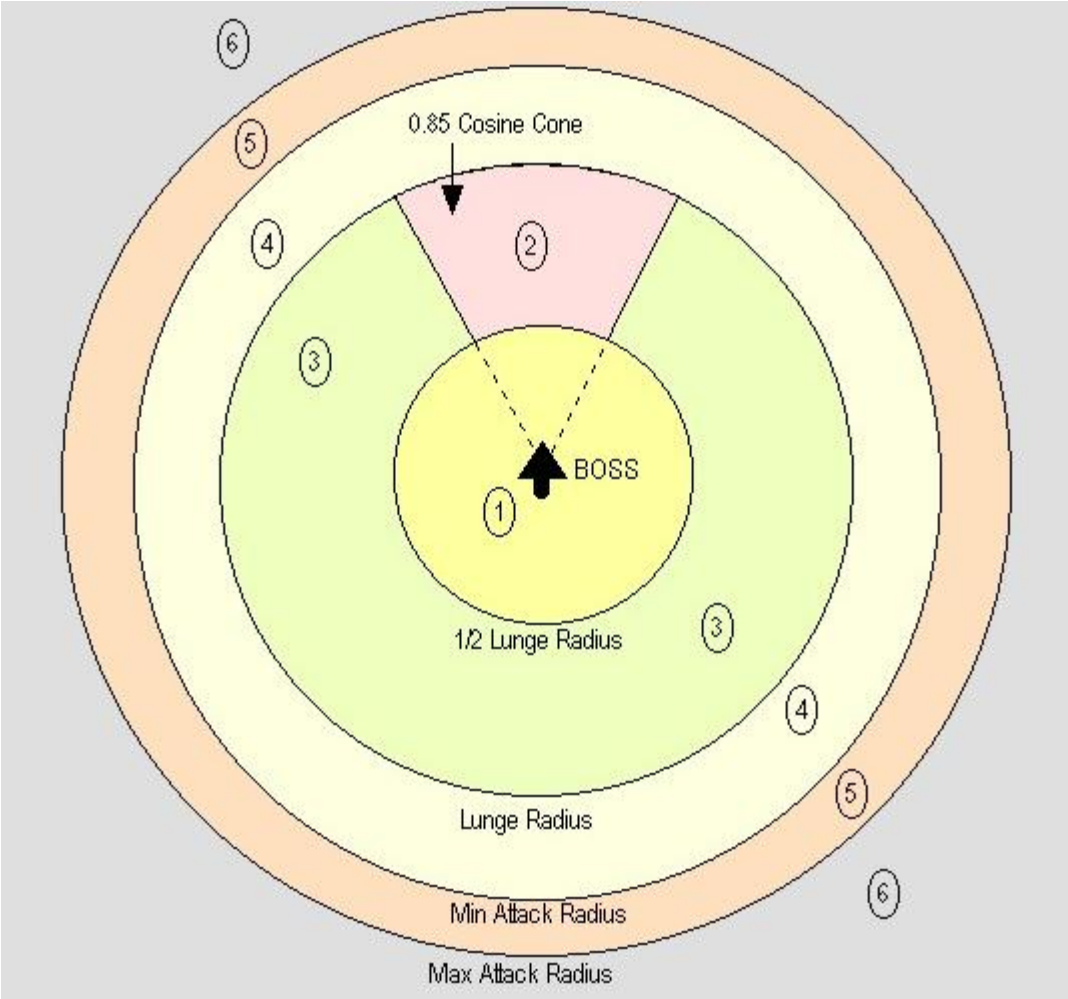
This boss starts with a shield, which uses the StartWithShield common boss behavior (`shieldContainer`). The boss typically has shields on, only temporarily turning them off when it fires. The shield is set to be always visible.

4.2.3.3 Weaponry

Each of the eight spikes has an invisible turret of type `EXC03_TURRET_PHOTON` (in turret.r). Each turret contains one charging photon gun (in weapon.r).

4.2.3.4 Behaviors Triggered by Player Position

Note: The boss's behaviors are being revised, and the diagram below will be updated when they're finalized.



The position of the player triggers various behaviors of the boss:

1	Yellow Zone: If the player is within one-half the lunge radius of the boss , the boss flees and fires (100% chance).	
2	Pink Zone: If the player is within the lunge radius and the 0.85 cosine angle cone but is further than half the lunge radius, the boss chooses:	
		Spike attack (100% chance if the photon guns are charged).
		To turn towards the player (50% chance if the photon guns are not charged).
		To chase the player (50% chance if the photon guns are not charged).
3	Light Yellow Zone: If the player is within the lunge radius but is further than half the lunge radius and is outside the 0.85 cosine angle cone, the boss randomly chooses:	
		To turn towards the player (50% chance).
		To chase the player (50% chance).

4	Pale Yellow Zone (same behaviors as #3): If the player is within the minimum attack radius but is further than the lunge radius, the boss randomly chooses:	
		To turn towards the player (50% chance).
		To chase the player (50% chance).
5	Orange Zone: If the player is with the maximum attack radius but is further than the minimum attack radius, the boss lunges at the player (100% chance).	
6	Gray Zone: If the player is outside the maximum attack radius, the boss moves toward the player (100% chance).	

4.2.3.5 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.3.6 Spike Attack

```
AIDataValue ( "Attack",         Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "Recharge",       Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "Rest",           Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "Breath",         Resource, ( Val = ( rtSequence, <resource ID> )))
```

When firing its photon gun spikes, Spike uses sequences that move its spikes forward, fire, move them back again, and so on:

- **Attack** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss makes a spike attack. The sequence (see the example below) plays an `rtMeshMorphAction` action in which you can change the speed at which the spikes move.
- **Recharge** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss recharges its photon guns.
- **Rest** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss is at rest.
- **Breath** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss breathes.

Example:

```
AIDataValue ( "Attack",         Resource, ( Val = ( rtSequence, EXC03_SEQ_MORPH_ATTACK )))
AIDataValue ( "Recharge",       Resource, ( Val = ( rtSequence, EXC03_SEQ_MORPH_RECHARGE
                  )))
AIDataValue ( "Rest",           Resource, ( Val = ( rtSequence, EXC03_SEQ_MORPH_REST )))
AIDataValue ( "Breath",         Resource, ( Val = ( rtSequence, EXC03_SEQ_MORPH_BREATH )))

defres rtSequence ( EXC03_SEQ_MORPH_ATTACK )
(
   Actions[] = ( ActionId = ( rtMeshMorphAction, EXC03_ACT_MORPH_ATTACK ) Time = 0 )
)

defres rtMeshMorphAction ( EXC03_ACT_MORPH_ATTACK )
(
   ModelFile     = "monster.dat"
```

```
   HierarchyName = "morph_joint"
   AnimType      = kemPlayThu
   Array[]       = ( ModelName = "morphhead_target01" Duration = 0.5 )
   Array[]       = ( ModelName = "morphhead_target02" Duration = 0.5 )
   Array[]       = ( ModelName = "morphhead_target03" Duration = 0.5 )
   Array[]       = ( ModelName = "morphhead_target04" Duration = 0.5 )
)
```

4.2.3.7 Chasing

```
AIDataValue ( "MaxChaseTicks",   Integer,   ( Val = <#> ))
```

Spike can chase the player for up to a specified set of time:

- **MaxChaseTicks** (integer; default is 50) specifies the maximum time (in ticks; 1 tick = 0.05 seconds; 1 second = 20 ticks) the boss chases the player before choosing another behavior.

Example:

```
AIDataValue ( "MaxChaseTicks",   Integer,   ( Val = 50 ))
```

4.2.3.8 Flee & Fire

If the player comes too close to the boss, it flees (using the Fleeing common boss behavior, `fleeDistance`), turns around, and fires all its guns. This is hardcoded with no tunable parameters. While fleeing, the boss has its effort level set to 3.0.

4.2.3.9 Getting Some Distance

Note: This behavior and its parameter are being hooked up into the boss as part of its revised behaviors.

```
AIDataValue ( "RunAwayDistanceGoal",   Float,   ( Val = <#> ))
```

The boss occasionally tries to get some distance from the player. It sets its effort level to 2.5 while doing this and uses:

- **RunAwayDistanceGoal** (floating point; default it 1200.0) specifies the distance (in meters) from the player the boss moves to. (When it decides to gets some distance, it picks a point `RunAwayDistanceGoal` from the player's current position and moves there.)

Example:

```
AIDataValue ( "RunAwayDistanceGoal",   Float,   ( Val = 1200.0 ))
```

4.2.3.10 Lunging

The boss can lunge at the player using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2.4 No EXCESSION_04

Level 4 does not have a boss.

4.2.5 EXCESSION_05 “Fan”

- *Klown Code File:* exc05.fsm
- *Klown:* DEvil_05
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, Attached, SetTarget, FireControl, Weapons, SearchForTarget, ExcAlive, ExcSound, Scorable, Skeleton, SelectiveDamage, StartWithShield, Master, TargetOffset
  —*Has Internal Klowns:* BossWings, NormalBehavior

The behaviors for this boss are:

- Has selective damage; can only be killed on the weak point behind the wings.
- Moves to different way points, activates shields while traveling between the waypoints. When shields are up, the two laser-armed turrets cannot fire but the single missile-armed turret is active.
- Circles and lays a clutch eggs once it arrives at a waypoint.
- After laying eggs, converts workers into warriors if there are any workers near the waypoint.
- Occasionally lays a single egg.
- Lays a maximum of 30 “turret” eggs (hardcoded).
- Makes turret attacks: runs away if player is too close, turns, and fires turrets at the player (raises wings and points forearms at him).

4.2.5.1 Selective Damage

The boss has a selective damage point behind its wings and can only killed on the weak point there. This uses the Selective Damage common boss behavior (`jointSelectiveDamage`). *Note:* Its wings open when it makes a turret attack, and the length of time the wings are open for this is controlled by a parameter. See Turret Attack.

4.2.5.2 Shields

This boss starts with a shield, which uses the StartWithShield common boss behavior (`shieldContainer`). The boss turns shields on as it moves to the next waypoint, and turns shields off upon arrival. When shields are up, the two laser-armed turrets cannot fire but the single missile-armed turret is active.

4.2.5.3 Weaponry

The boss has two energy-weapon turrets and one missile-firing turret.

4.2.5.4 Behavior Hierarchy

Once the boss arrives on the level and does its initial tasks (like ordering workers to go to waypoints), the boss's behaviors are governed by the following hierarchy:

IF the boss has been at a waypoint for more than 60 seconds:

THEN travel to its next waypoint (see Traveling to Waypoints).

ELSE IF the boss hasn't laid all its eggs at the waypoint yet:

THEN keep laying eggs one at a time until all are laid (see Laying Eggs at Waypoints).

ELSE IF there are any workers near the waypoint:

THEN convert them workers (see Converting Workers to Warriors).

ELSE IF either the player is more than 16 boss radii away or on a 50% chance"

THEN lay one egg (see Circle & Lay Eggs).

ELSE fire turrets on player (see Turret Attack).

4.2.5.5 Traveling to Waypoints

```
AIDataValue ( "waypoints",           Resource, ( Val = ( rtVectorArray, <resource ID> )))
AIDataValue ( "waypointSpeed",       Float,    ( Val = <#> ))
AIDataValue ( "maxDistanceFromHome", Float,    ( Val = <#> ))
```

The boss can travel to specified waypoints at a set speed, pausing at each waypoint for 60 seconds (this value currently is not tunable). While pausing at a waypoint, it can move in the vicinity of the waypoint up to a maximum distance from it, and perform various actions there until it heads off for the next waypoint. These behaviors are controlled by the following parameters:

- **waypoints** (resource; no default) sets the waypoints to which the boss travels. It points to an `rtVectorArray` resource named *<resource ID>*, which is the list of the waypoints (in world coordinates) for the boss. There can be any number of waypoints in the list. When the boss reaches the last waypoint in the list, it starts over again, traveling to the first waypoint. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtVectorArray` resources.
- **waypointSpeed** (floating point; default is 1.0) sets the effort level (the speed) the boss uses to move between waypoints. Enter a floating point value for *<#>*; this is a multiplier value that is applied to the standard motion parameters described in the `rtBodyMotion` and `rtBodyControl` resources for the boss.
- **maxDistanceFromHome** uses the Maximum Distance from Home common boss behavior and controls the maximum distance the boss moves from a waypoint while pausing there. (The boss is hardcoded to reset its home position to its current waypoint.) Enter a floating point value for *<#>*; this is maximum distance in meters that the boss moves from the waypoint.

Example:

```
AIDataValue ( "waypoints",           Resource, ( Val = ( rtVectorArray, EXCESSION_05 )))
AIDataValue ( "waypointSpeed",       Float,    ( Val =    5.0 ))
AIDataValue ( "maxDistanceFromHome", Float,    ( Val = 1200.0 ))

defres rtVectorArray ( EXCESSION_05 )
```

```
(
   Vec[] = ( Value = (     0, -2000, 0 ))
   Vec[] = ( Value = (  1500,  1200, 0 ))
   Vec[] = ( Value = ( -1500,  1200, 0 ))
)
```

4.2.5.6 Laying Eggs at Waypoints

AIDataValue ("eggsToLayAtWaypoint", Integer, (Val = <#>))

The boss is hardwired to lay eggs immediately after it arrives at a waypoint, laying one at a time until it has laid all its eggs for the waypoint:

- **eggsToLayAtWaypoint** (integer; default is 3) sets the number of eggs the boss lays at the waypoint.

Example:

```
AIDataValue ( "eggsToLayAtWaypoint", Integer, ( Val = 4 ))
```

The weapon EXW05_EGGLAYER (in missile.r) contains the weapon used by the boss to lay the eggs. The weapon must use the GUNCAT_MINES gun category. The firing sequence contains the mesh morph for the boss and describes what egg is laid. The egg EXS_EGG has an rtSequence resource that specifies how long the egg takes to hatch and what kind of enemy hatches from the egg. (Note that the rtEmitInfo action for the hatchling starts only after the model scaling action for the egg is complete. Changing the times for the model scaling action thus changes the incubation time for the egg.)

Example:

```
defres rtSequence ( EXS_EGG )
(
   Actions[] = ( ActionId = ( rtMeshMorphAction,    EXS_EGG ) Time = 0 )
   Actions[] = ( ActionId = ( rtModelScalingAction, EXS_EGG ) Time = 0 Ref = 1)

   // These two should start at the same time...
   Actions[] = ( ActionId = ( rtAlphaAction, EXS_EGG ) Time = 500 After = 1 Ref = 2 )
   Actions[] = ( ActionId = ( rtEmitInfo,    EXS_EGG ) Time = 500 After = 1 )

   Actions[] = ( ActionId = ( rtDeathAction, EXS_EGG ) Time = 0 After = 2 )
)

defres rtModelScalingAction ( EXS_EGG )
(
   AnimiType = kemPlayThu
   Array[]   = ( SizeVec = ( 1.5, 1.5, 1.5 ) IntervalDuration = 3000 )
   Array[]   = ( SizeVec = ( 2.5, 2.5, 2.5 ) IntervalDuration = 7500 )
   Array[]   = ( SizeVec = ( 3.0, 3.0, 3.0 ) IntervalDuration = 3000 )
)
```

The egg itself uses an “Entity” klown, which does nothing but accept damage. The health of the egg can be set in the egg’s rtAI property. See Egg for more details.

4.2.5.7 Circle & Lay Eggs

This behavior uses the Circling the Player or a Location common boss behavior (`circleDistance`) and lays a single egg (per laying eggs at waypoints, except only one egg is laid).

4.2.5.8 Turret Attack

```
AIDataValue ( "turretAttackSequence",   Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "turretAttackTime",       Float,    ( Val = <#> ))
```

When the boss starts a turret attack, if the player is within eight boss radii, the boss first flees, using the Fleeing common boss behavior (`fleeDistance`), with its effort level set to 2.5 while fleeing. This opens up some distance between the boss and the player before the boss fires it turrets. (If the player is beyond eight boss radii, the boss does not flee before firing).

The boss turns toward the player and fires its turrets. It also uses the following parameters:

- **`turretAttackSequence`** (resource; no default) specifies the articulation sequence that, once the boss is far enough from the player, causes the boss to raise its wings and arms and fire its turrets at the player. This sequence of course must be defined as part of the boss's resources.
- **`turretAttackTime`** (floating point; default is 8.0) specifies the amount of time (in seconds) the boss's wings are open when making a turret attack. (When the wings are open, the boss's selective damage point is exposed.)

When making its attack, the boss activates all attached turrets that are labeled `Tag = 0`. After the attack, these turrets are deactivated.

Note: For the above articulation sequence to work correctly, you must also use the `disableSkeletalPhysics` parameter (see Disabling Skeletal Physics). Otherwise, the skeletal spring system fights against the articulation.

Example:

```
AIDataValue ( "turretAttackSequence",   Resource, ( Val = ( rtSequence,
                EXC05_SEQ_TURRETATTACK )))
AIDataValue ( "disableSkeletalPhysics", Integer,  ( Val = 1 ))
AIDataValue ( "turretAttackTime",       Float,    ( Val = 2.0 ))
```

4.2.5.9 Converting Workers to Warriors

```
AIDataValue ( "acceptableRNATargets", Integer,  ( Val = <object type> ))
AIDataValue ( "RNASequence",          Resource, ( Val = ( rtSequence, <resource ID> )))
```

When the boss arrives on the level, the first thing it does is to command the workers to go to the nearest waypoint, around which they will orbit. A worker's orbit radius is hardcoded to be eight times the radius of the worker.

After the boss arrives at a waypoint and lays its eggs, if it detects a worker within 1000m (currently hardcoded), it tries to convert the worker. It activates its attached turrets labeled with `Tag = 2` (this is the `EXC05_TURRET_EGG` turret and is defined in turret.r). This turret works like a normal turret, with one exception: Its gun (`EXW_EGGFROMWORKER` in missile.r) uses a shot (`EXS_EGGFROMWORKERSHOT` in missile.r) that uses Playing Sequences on Targets to play a sequence on the target.

- **`acceptableRNATargets`** specifies all of the valid targets on which the `RNASequence` is played if the shot hits the target. Use `kSenseWorker` to specify workers as valid targets.

- **RNASequence** plays an `rtSequence` resource named *<resource ID>* on the target. For the boss, the `EXS_EGGFROMWORKERSHOT_SEQ1` sequence removes the worker (or other valid target) and emits a warrior egg (`EXS_WARRIOR_EGG` in missile.r) in the worker's place. The warrior egg works exactly like the eggs described above in the Egg Laying section (namely, it has an `rtSequence` resource that grows the egg and emits a hatchling).

Example:

```
AIDataValue ( "acceptableRNATargets", Integer,  ( Val = kSenseWorker ))
AIDataValue ( "RNASequence",          Resource, ( Val = ( rtSequence,
                EXS_EGGFROMWORKERSHOT_SEQ1 )))
```

4.2.5.10 Reaction to Sinibombs

If the boss takes damage from a Sinibomb, it immediately raises its shield and begins to travel to its next waypoint, regardless of whatever it was doing at that point.

4.2.6 EXCESSION_06 “Larva” (“Space Hen”)

- *Klown Code File:* exc06.fsm
- *Klown:* DEvil_06
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, Avoidance, SphericalRestriction, SetTarget, SearchForTarget, Weapons, TargetOffset, ExcAlive, ExcSound, ExcPain, StrikingArms, Exc06_WhippingTail, TractorBeam2, TractorAntiMissile, Attached
  —*Has Internal Klowns:* NormalBehavior

The behaviors for this boss are:

- Reacts to extreme pain (low health) by speeding up.
- Has head, arm, and tail articulations.
- Has a tractor beam, which it can use to tractor in the player or to repel the player and Sinibombs.
- Can tractor and whack, repel, fire missiles at, fire its tail at, makes roly poly attacks on, circle, and retreat from the player, based on the player’s position.

4.2.6.1 Pain Reaction

The boss uses the common pain reaction parameters (`painReaction`, `painReactionTime`; see Common Articulation Parameters). In addition, if the boss’s health goes below 30% of its maximum health value, the boss’s default effort level is set to 1.5 (instead of 1.0).

4.2.6.2 Head, Arm, & Tail Articulations

- *Klown Code File:* whippingtail.fsm
- *Klown:* Exc06_WhippingTail
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

Note: This boss reuses the Head Sweeping and Striking Arms parameters from the common behaviors, as well as having some new parameters for its tail. However, these parameters are controlled by a different klown and can have different defaults than the common parameters. Accordingly, all the parameter of the boss’s articulation klown are included here. New material is shown in black text; material repeated from Common Articulation Parameters are shown in gray text, which any changes to it shown in red text.

4.2.6.2.a Joint Resistance and Damping

```
AIDataValue ( "jointResistance",    Float,    ( Val = <#> ))
AIDataValue ( "jointDamping",       Float,    ( Val = <#> ))
```

The head, arms, and tail of the boss can articulate. Articulation requires two general parameters:

- **`jointResistance`** (floating point; default is 200.0) specifies the stiffness of the movement and can be equal to or greater than 0.0. Values below 10.0 make the movement very loose and values above 90.0 make it very stiff (and at or above 100.0 the joint becomes too stiff to move).
- **`jointDamping`** (floating point; >= 0.0 to <= 1.0; default is 0.9) specifies the damping factor, which is the time in seconds the articulation movement takes to come to rest.

Example:

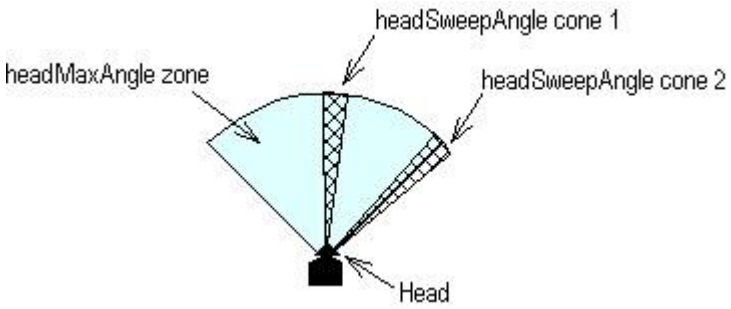
```
AIDataValue ( "jointResistance",  Float, ( Val = 200.0 ))
AIDataValue ( "jointDamping",     Float, ( Val =   0.9 )))
```

4.2.6.2.b Head Sweeping Parameters

```
AIDataValue ( "headMaxAngle",        Float,    ( Val = <#> ))
AIDataValue ( "headSweepAngle",      Float,    ( Val = <#> ))
AIDataValue ( "headSweepSpeed",      Float,    ( Val = <#> ))
AIDataValue ( "headTargetOffset",    Vector,   ( Val = ( <#>, <#>, <#> )))
```

The boss can articulate its head , sweeping it and striking targets with it:

- **headMaxAngle** (floating point; >= -1.0 to <= 1.0; default is 0.8) specifies the cosine of the angle (see Quick Table of Cosines) that defines the head's maximum turning zone. The head can turn to "look at" its target only within this zone.
- **headSweepAngle** (floating point; >= -1.0 to <= 1.0; default is 0.2) specifies the cosine of the angle that defines the head's actual sweep cone for a specific sweep. The normal of the cone is whatever direction the head is currently facing in its `headMaxAngle`. While the head can sweep anywhere in its `headMaxAngle` maximum sweep zone, a single sweep can move only in its more limited `headSweepAngle` cone. Note that if the `headSweepAngle` cone extends beyond the `headMaxAngle` (see diagram), the head still will not sweep outside its `headMaxAngle` zone. A head can sweep through the entire volume of Cone 1, while it could only sweep through the part of Cone 2 that lies within the `headMaxAngle` zone. (If the head is turned to its full `headMaxAngle`, then the head can only sweep through half its `headSweepAngle` cone.)



- **headSweepSpeed** (floating point; default is 70.0) specifies the speed of the head's sweep (higher values mean faster sweeps).
- **headTargetOffset** (vector; default is 0.0, 12.0, 0.0) specifies the offset for the head at which it sweeps.

Example:

```
AIDataValue ( "headMaxAngle",        Float,    ( Val =   0.8 ))
AIDataValue ( "headSweepAngle",      Float,    ( Val =   0.0 ))
AIDataValue ( "headSweepSpeed",      Float,    ( Val = 100.0  ))
AIDataValue ( "headTargetOffset",    Vector,   ( Val = ( 0.0, 0.0, 25.0 )))
```

4.2.6.2.c Striking Arm Parameters

The boss ***does*** use the Striking Arms parameters (`armMaxAngle`, etc.) of the klown in strikingarm.fsm, and accordingly these parameters are not repeated here. However, the Exc06_WhippingTail klown also contains two arm parameters:

```
AIDataValue ( "armResistance",       Float,    ( Val = <#> ))
AIDataValue ( "armDamping",          Float,    ( Val = <#> ))
```

These parameters control the resistance and damping of the arms, separate from the common joint resistance and damping parameters:

- **armResistance** (floating point; default is 50.0) specifies the stiffness of the arm's movement.
- **armDamping** (floating point; >= 0.0 to <= 1.0; default is 0.4) specifies the arm's damping factor, which is the time in seconds the articulation movement takes to come to rest.

Example:

```
AIDataValue ( "armResistance",     Float, ( Val = 50.0 ))
AIDataValue ( "armDamping",        Float, ( Val =  0.4 )))
```

4.2.6.2.d Whipping Tail Parameters

```
AIDataValue ( "whipTailOutSeq",  Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "whipTailBackSeq", Resource, ( Val = ( rtSequence, <resource ID> )))
```

The following parameters control the whipping of the boss's tail:

- **whipTailOutSeq** (resource; no default) specifies the `rtSequence` resource that plays when the boss whips its tail out.
- **whipTailBackSeq** (resource; no default) specifies the `rtSequence` resource that plays when the boss whips its tail back.

Example:

```
AIDataValue ( "whipTailOutSeq",  Resource, ( Val = ( rtSequence, EXC06_TAIL_OUT_SEQ )))
AIDataValue ( "whipTailBackSeq", Resource, ( Val = ( rtSequence, EXC06_TAIL_BACK_SEQ )))
```

4.2.6.3 Weaponry

- *Klown Code File:* whippingtail.fsm
- *Klown:* Exc06_WhippingTail
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
AIDataValue ( "backWeapon",      Integer,  ( Val = <resource ID> ))
AIDataValue ( "tailWeapon",      Integer,  ( Val = <resource ID> ))
```

The boss has four concussion bursts turrets, a missile gun on its back, and another missile gun in its tail. The boss also has an anti-Sinibomb weapon. The boss has parameters for the back and tail weapons:

- **backWeapon** (resource ID; default is -1) specifies the resource ID of the back's weapon category. If the default is used, then the boss behaves as if it has no back weapon.
- **tailWeapon** (resource ID; default is -1) specifies the resource ID of the tail's weapon category. If the default is used, then the boss behaves as if it has no tail weapon. (This parameter is not part of the DEvil_06 klown in exc06.fsm, but instead appears in the Exc06_WhippingTail klown of whippingtail.fsm.)

Example:

```
AIDataValue ( "tailWeapon",      Integer,  ( Val = GUNCAT_MAIN ))
AIDataValue ( "backWeapon",      Integer,  ( Val = GUNCAT_MISSILE ))
```

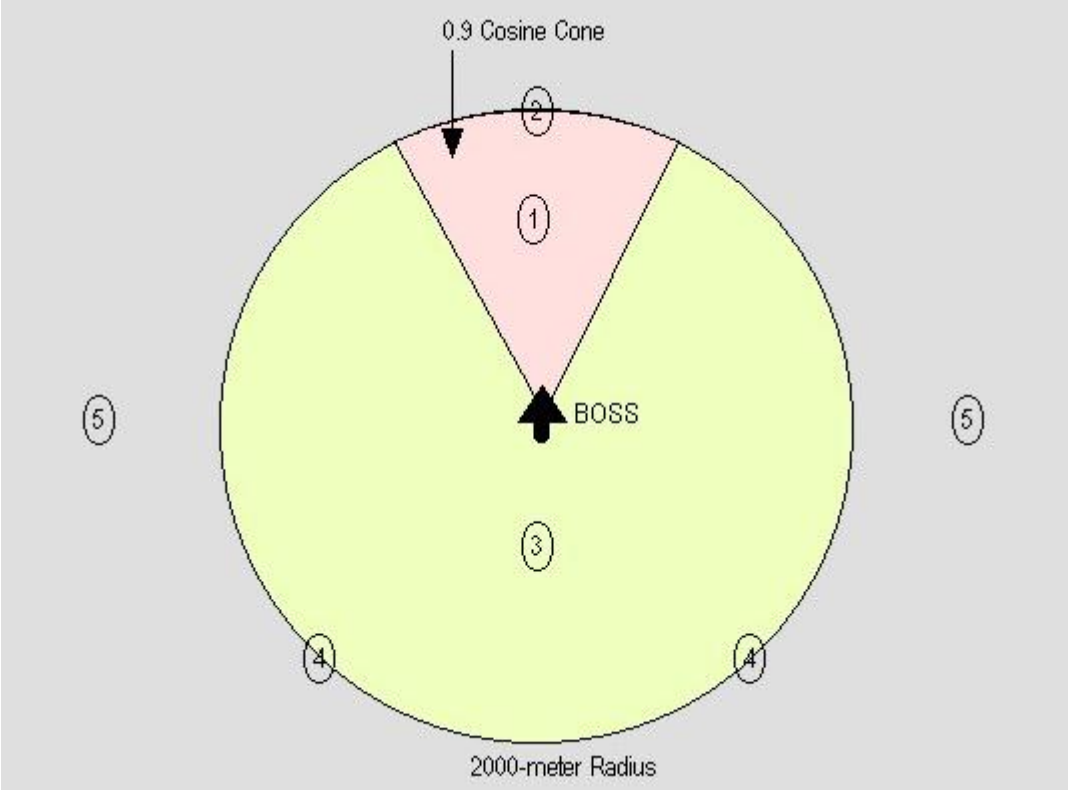
4.2.6.4 Tractor Beam

The boss uses the Tractor Beam 2 parameters for its tractor beam (`tractorBeam` and up to 21 more parameters).

4.2.6.5 Anti-Sinibomb Defenses

This boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCategory`, `antiMissileCosAngle`) to detect incoming Sinibombs and use its tractor beam to repel them. (This works the same way as the tractor beam repelling the player; see Massive Repulsiveness.)

4.2.6.6 Behaviors Triggered by Player Position



Note: the 2000-meter radius and the 0.9 cosine cone angle are hardcoded.

The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is less than 2000 meters from the boss and is within the 0.9 cosine cone of the boss, the boss randomly chooses:	
		To tractor and whack (with weapons) the player (20% chance).
		To repel the player (20% chance).
		To attack the player with missiles (20% chance).
		To make a roly poly attack on the player (20% chance).
		To retreat from the player, firing missiles (20% chance).

2	Outer Boundary of the Red Zone: If the player is exactly 2000 meters from the boss and is within the 0.9 cosine cone of the boss, the boss randomly chooses:	
		To tractor the player (20% chance).
		To repel the player (20% chance).
		To attack the player with missiles (20% chance).
		To circle the player (20% chance).
		To retreat from the player, firing missiles (20% chance).
3	Yellow Zone: If the player is less than 2000 meters from the boss and is outside the 0.9 cosine cone, the boss randomly chooses:	
		To circle the player (40% chance).
		To attack the player with missiles (20% chance).
		To make a roly poly attack on the player (20% chance).
		To retreat from the player, firing missiles (20% chance).
4	Outer Boundary of the Zone: If the player is exactly 2000 meters from the boss and is within the 0.9 cosine cone, the boss randomly chooses:	
		To circle the player (60% chance).
		To attack the player with missiles (20% chance).
		To retreat from the player, firing missiles (20% chance).
5	Gray Zone: If the player is more than 2000 meters from the boss, the boss randomly chooses:	
		To circle the player (40% chance).
		To attack the player with missiles (20% chance).
		To attack the player with its tail (20% chance).
		To retreat from the player, firing missiles (20% chance).

4.2.6.7 Tractor & Whack Attack

```
AIDataValue ( "whackTime",          Float,    ( Val = <#> ))
```

This boss grabs the player in its tractor beam, draws the player in to a hardcoded distance, and whacks the player (by firing its weapons on the player). The time the boss spends whacking the player is controlled by:

- **whackTime** (floating point; no default) specifies the amount of time (in seconds) that the boss keeps the player held in position while firing weapons on the player.

Example:

```
AIDataValue ( "whackTime",          Float,    ( Val = 3.0 ))
```

4.2.6.8 Massive Repulsiveness

```
AIDataValue ( "repelTime",          Float,    ( Val = <#> ))
```

This boss grabs the player in its tractor beam and pushes the player away for a length of time:

- **repelTime** (floating point; no default) specifies the amount of time (in seconds) that the boss pushes the player away.

Example:

```
AIDataValue ( "repelTime",          Float,    ( Val = 3.0 ))
```

4.2.6.9 Circling

The boss can attempt to circle the player, using the Circling the Player or a Location common boss behavior (`circleDistance`).

4.2.6.10 Roly Poly Attack

The boss first rolls over and fires its concussion turrets, and then it rolls on its back and fires its missiles. The boss does this for three seconds (hardcoded) when making a roly poly attack.

4.2.6.11 Tail Attack

The boss aims at the player from a long way off (the player must be more than 2000 meters away) and fires its tail.

4.2.6.12 Retreating

When the boss retreats, it flees for 10 seconds (using the Fleeing common boss behavior), firing missiles from its back as it flees. The boss has its effort level set to 2.0 while fleeing.

4.2.6.13 Smash & Squish

```
AIDataValue ( "smashTime",          Float,    ( Val = <#> ))
AIDataValue ( "squishTime",         Float,    ( Val = <#> ))
```

At one point, the boss was to have the ability to use its tractor beam to smash the player against asteroids and to squish asteroids against the player. These behaviors are not currently operational, but two parameters for them are exposed in the code:

- **smashTime** (floating point; no default).
- **squishTime** (floating point; no default).

Example:

```
AIDataValue ( "smashTime",          Float,    ( Val = 3.0 ))
AIDataValue ( "squishTime",         Float,    ( Val = 3.0 ))
```

4.2.7 EXCESSION_07 “Grinder”

- *Klown Code File:* exc07.fsm
- *Klown:* DEvil_07
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SphericalRestriction, SetTarget, Weapons, SearchForTarget, TargetOffset, ExcAlive, ExcSound, Scorable, Exc07_GrindingDrills, Exc07_PowerSack, ExcPain, ForceField, TractorAntiMissile
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has a tractor beam, which it can use to tractor in asteroids and the player or to repel the player and Sinibombs.
- Has drills, which can smash up asteroids and the player.
- Has a crystal magnet, to draw in mined crystals.
- Has a power sack, to store crystal energy and to pulse it at the player.

4.2.7.1 Pain Reaction

The boss uses the common pain reaction parameters (`painReaction`, `painReactionTime`; see Common Articulation Parameters).

4.2.7.2 The Drills that Grind; the Bits that Bite

- *Klown Code File:* grindingdrills.fsm
- *Klown:* Exc07_GrindingDrills
  —*Type:* Submachine
  —*Uses External Klown:* Skeleton

```
   // Drill Sequences
AIDataValue ( "drillIdleSeq",        Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "drillFastSeq",        Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "drillSpeedUpSeq",     Resource, ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "drillSlowDownSeq",    Resource, ( Val = ( rtSequence, <resource ID> ))

   // Drill Operation Parameters
AIDataValue ( "drillIdleSpeed",      Float,    ( Val = <#> ))
AIDataValue ( "drillTopSpeed",       Float,    ( Val = <#> ))
AIDataValue ( "drillSpinUpTime",     Float,    ( Val = <#> ))
AIDataValue ( "drillSpinDownTime",   Float,    ( Val = <#> ))
AIDataValue ( "drillDamage",         Float,    ( Val = <#> ))
```

4.2.7.2.a Drill Sequences

Grinder plays the following sequences on its drills:

- **`drillIdleSeq`** (resource; no default) specifies the resource ID of the sequence plays when the drills are idling.
- **`drillFastSeq`** (resource; no default) specifies the resource ID of the sequence plays when the drills are at their maximum speed.

- **drillSpeedUpSeq** (resource; no default) specifies the resource ID of the sequence plays when the drills are speeding up to their maximum speed.
- **drillSlowDownSeq** (resource; no default) specifies the resource ID of the sequence plays when the drills are slowing down to their idle speed.

Example:

```
AIDataValue ( "drillIdleSeq",      Resource, ( Val = ( rtSequence, EXC07_SEQ_SND_IDLE ))) 
AIDataValue ( "drillFastSeq",      Resource, ( Val = ( rtSequence, EXC07_SEQ_SND_FAST ))) 
AIDataValue ( "drillSpeedUpSeq",   Resource, ( Val = ( rtSequence, EXC07_SEQ_SND_SPEEDUP 
                ))) 
AIDataValue ( "drillSpeedDownSeq", Resource, ( Val = ( rtSequence, EXC07_SEQ_SND_SLOWDOWN 
                )))
```

4.2.7.2.b Drill Operation Parameters

Grinder has a set of drills that can grind asteroids and the player, damaging them in the process. The drills are turned on when Grinder is ready to grind and turned off when Grinder is done. As Grinder grinds, the drills vary in speed. The drills are controlled by:

- **drillIdleSpeed** (floating point; default is 1.0) specifies the speed of the drills when they are on but not actually grinding something.
- **drillMaxSpeed** (floating point; default is 5.0) specifies the maximum speed the drills can attain when they are grinding something.
- **drillSpinUpTime** (floating point; default is 2.0) specifies the time (in seconds) it takes the drills to go from idle speed to maximum speed.
- **drillSpinDownTime** (floating point; default is 5.0) specifies the time (in seconds) it takes the drills to go from maximum speed to idle speed.
- **drillDamage** (floating point; default is 1.0) is a multiplicative factor used with the drill's current speed to determine the amount of damage is applied to an object being ground:

  Damage = drillDamage * Drill Current Speed

Example:

```
AIDataValue ( "drillIdleSpeed",      Float,    ( Val = 1.0 )) 
AIDataValue ( "drillTopSpeed",       Float,    ( Val = 5.0 )) 
AIDataValue ( "drillSpinUpTime",     Float,    ( Val = 2.0 )) 
AIDataValue ( "drillSpinDownTime",   Float,    ( Val = 5.0 )) 
AIDataValue ( "drillDamage",         Float,    ( Val = 1.0 ))
```

4.2.7.3 Crystal Magnet

```
AIDataValue ( "roidSensor",   Resource,   ( Val = ( rtBodySensor, <resource ID> ))) 
AIDataValue ( "maxForce",     Float,      ( Val = <#> )) 
AIDataValue ( "minForce",     Float,      ( Val = <#> ))
```

Grinder's crystal magnet imparts a force to crystals within range of the boss's crystal sensor, which propels the crystals towards the boss. The force applied depends upon the distance from the boss to the crystal and is linearly scaled between its maximum and minimum values:

- **roidSensor** (resource; default is no sensor) specifies the sensor Grinder uses to detect asteroids. See Sensors for more details on sensors.

- **maxForce** (floating point; no default) specifies the maximum attractive force applied to crystals. This force is used for detected crystals closest to the boss.
- **minForce** (floating point; no default) specifies the minimum attractive force applied to crystals. This force is used for detected crystals farthest from the boss.

Note: `maxForce` and `minForce` should be negative values with the absolute value of `maxForce` greater than the absolute value of `minForce`.

Example:

```
AIDataValue ( "roidSensor",   Resource,   ( Val = ( rtBodySensor, EXC07_ROID_SENSOR )))
AIDataValue ( "maxForce",     Float,      ( Val = -5000.0 ))
AIDataValue ( "minForce",     Float,      ( Val =  -500.0 ))
```

4.2.7.4 Power Sack

- *Klown Code File:* grindingdrills.fsm
- *Klown:* Exc07_PowerSack
  —*Type:* Submachine

```
AIDataValue ( "sackMaxPower",    Float,    ( Val = <#> ))
AIDataValue ( "sackInitPower",   Float,    ( Val = <#> ))
AIDataValue ( "sackMinScale",    Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "sackMaxScale",    Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "sackGrowRate",    Float,    ( Val = <#> ))
```

Grinder has a power sack in which it stores the energy from the crystals it mines. The sack grows and shrinks as power pulses into and out of the sack. The sack is scaled between its minimum and maximum possible sizes proportionally based on its current power factor (current crystal power divided by the maximum power the sack can hold). The following parameters control the sack:

- **sackMaxPower** (floating point; default is 100.0) specifies the maximum crystal power that can be stored in the sack.
- **sackInitPower** (floating point; default is 0.0) specifies the crystal power that the sack starts with.
- **sackMinScale** (vector; default is 1.0, 1.0, 0.5) specifies the minimum scale of the sack (the size of the sack when it has no power).
- **sackMaxScale** (vector; default is 2.0, 2.0, 2.0) specifies the maximum scale of the sack (the size of the sack when it is at `sackMaxPower`).
- **sackGrowRate** (floating point; >= 0.0 to <=1.0; default is 0.1) specifies rate per second at which the sack can grow or shrink. For example, `Val = 1.0` means the sack can grow or shrink to 100% of its scaling change in one second.

Example:

```
AIDataValue ( "sackMaxPower",    Float,    ( Val =  50.0 ))
AIDataValue ( "sackInitPower",   Float,    ( Val =   0.0 ))
AIDataValue ( "sackMinScale",    Vector,   ( Val = ( 1.0, 1.0, 0.5 )))
AIDataValue ( "sackMaxScale",    Vector,   ( Val = ( 2.0, 2.0, 2.0 )))
AIDataValue ( "sackGrowRate",    Float,    ( Val =   0.5 ))
```

4.2.7.5 Tractor Beam

The boss uses the Tractor Beam 2 parameters for its tractor beam (`tractorBeam` and up to 21 more parameters).

4.2.7.6 Anti-Sinibomb Defenses

This boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCosAngle`) to detect incoming Sinibombs and use its tractor beam to repel them.

4.2.7.7 Behavior Summary

Grinder starts in defensive posture and moves between defensive and offensive postures based on the status of its power sack. When in a posture, it chooses a behavior based on its current condition:

Defensive Posture	If the player is inside one half the `backAwayDistance` of the boss, the boss backs away.
	Otherwise, the boss looks for and mines asteroids.
	When the boss's crystal sack is full, the boss switches to offensive posture.
Offensive Posture	The boss can make a tractor & drill attack on the player.
	The boss can make a power sack attack on the player.
	When the boss's crystal sack becomes empty, the boss switches to defensive posture.

4.2.7.8 Backing Away

The boss uses the Back Away from Enemy common boss behavior (`backAwayDistance`) when backing away from the player.

4.2.7.9 Mining Asteroids

```
AIDataValue ( "maxMineDistance",   Float,   ( Val = <#> ))
AIDataValue ( "minMineDistance",   Float,   ( Val = <#> ))
```

As Grinder wanders about seeking asteroids to mine, it resets it home position and uses the following parameters:

- **`maxMineDistance`** (floating point; no default) specifies the maximum distance from the boss's home position at which the boss will select an asteroid to mine. This prevents the boss from deciding to mine an otherwise juicy asteroid that's too far away.
- **`minMineDistance`** (floating point; no default) specifies the minimum distance from the boss's home position at which the boss will select an asteroid to mine. This prevents possible problems arising from the boss deciding to mine an asteroid that is real close to it.

The boss selects a roid it can find that's within its `minMineDistance` and beyond its `minMineDistance`. (The boss's klown further selects a crystal-laden roid that is the best distance within the zone set by these two parameters. These aspects of the boss's behavior are hardcoded.

Example:

```
AIDataValue ( "maxMineDistance",   Float,   ( Val = 1500.0 ))
AIDataValue ( "minMineDistance",   Float,   ( Val =  600.0 ))
```

4.2.7.10 Tractor & Drill Attack

The boss uses its tractor beam to draw the player into its drills, which smash the player like they smash asteroids (i.e., inflict damage).

4.2.7.11 Power Sack Attack

The boss fires on the player, pulsing out energy from its power sack.

4.2.8 No EXCESSION_08

Level 8 does not have a boss.

4.2.9 EXCESSION_09 “Zap”

- *Klown Code File:* exc09.fsm
- *Klown:* DEvil_09
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, Avoidance, SetTarget, Boss09MainWeapon, AntiMissileDefense, SearchForTarget, ExcAlive, ExcSound, Attached, Cloak, SKeleton, SelectiveDamage, TargetOffset
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has a cloak.
- Can wiggles and breathe.
- If the player is too far away, it moves or clumsily charges toward the player.
- If the player is close and the boss is almost in front or behind the player, the boss does one of: cloak, lunge, circle, zigzag.
- At other ranges to the player, it tries to cloak and chase the player.
- Has a selective damage point.

4.2.9.1 Selective Damage

The boss can only be killed at one point, which uses the Selective Damage common boss behavior (`jointSelectiveDamage`). Currently, this resource causes damage taken here to be doubled.

4.2.9.2 Articulations

```
AIDataValue ( "Wiggle",   Resource,   ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "Breath",   Resource,   ( Val = ( rtSequence, <resource ID> ))
```

This boss wiggles and breathes:

- **`Wiggle`** (resource; no default) specifies the `rtSequence` resource that plays when the boss wiggles.
- **`Breath`** (resource; no default) specifies the `rtSequence` resource that plays when the boss breathes.

Example:

```
AIDataValue ( "Breath",   Resource,   ( Val = ( rtSequence, EXC09_SEQ_MORPH_BREATH ))
AIDataValue ( "Wiggle",   Resource,   ( Val = ( rtSequence, EXC09_SEQ_WIGGLE ))
```

4.2.9.3 Cloak

```
AIDataValue ( "ContainerModel",      Integer,  ( Val = <resource ID> ))
```

AIDataValue ("CloakModel", Integer, (Val = *<resource ID>*))

This boss has a cloak, which uses the Energy-Consuming Cloaks common parameters of the Cloak klown (`CloakMaxEnergy`, `EnergyConsumptionPerTick`, `CloakAbortEnergy`, and `CloakMinEnergy`). The Cloak klown itself also calls on the ShipModel cloak, so that you can specify model-related cloaking parameters (`cloakModel`, `cloakFadeTime`). See Scorable Klown.

Example:

```
AIDataValue ( "CloakMaxEnergy",             Float,    ( Val = 100.0 ))
AIDataValue ( "EnergyConsumptionPerTick",   Float,    ( Val =   1.0 ))
AIDataValue ( "CloakAbortEnergy",           Float,    ( Val =  10.0 ))
AIDataValue ( "CloakMinEnergy",             Float,    ( Val =  30.0 ))
AIDataValue ( "firstPersonModel",           Integer,  ( Val = EXCESSION_09 ))
AIDataValue ( "cloakModel",                 Integer,  ( Val = CLOAK_EXCESSION_09 ))
AIDataValue ( "modelFadeTime",              Float,    ( Val = 3.0 ))
```

4.2.9.4 Lightning Gun

- *Klown Code File:* exc09.fsm
- *Klown:* Boss09MainWeapon
  —*Type:* CHFSM
  —*Uses External Klowns:* Weapons, ChargingMainWeapon, PollFast

The boss has a lightning gun, which fires lightning bolts and the player. The gun is `XW_EXC09_BOLT` (in lightning.r) and works along the same lines as the player lighting gun (see the lightning.r file and Player Weapons in the Sinistar design documentation). The gun recharges between firings (this uses the lightning gun's recharging ability, and not the Charging Main Weapon common boss behavior).

4.2.9.5 Anti-Sinibomb Defenses

This boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCategory`) to detect incoming Sinibombs and to fire its lightning gun at them.

4.2.9.6 Player Distance

AIDataValue ("Devil7_FarAway", Float, (Val = *<#>*))

When the player is too far away, the boss tries to move closer to the player by randomly (with equal probability) choosing to move or lunge toward the player.

If the player is within the boss's `lungeDistance` and the boss is in front of or behind the player, the boss randomly chooses one the following behaviors: Lunge (33% chance), Cloak (17%), Circle (16%), or Zigzag (34%). In any attack, the boss will fire its lightning gun when it gets a chance.

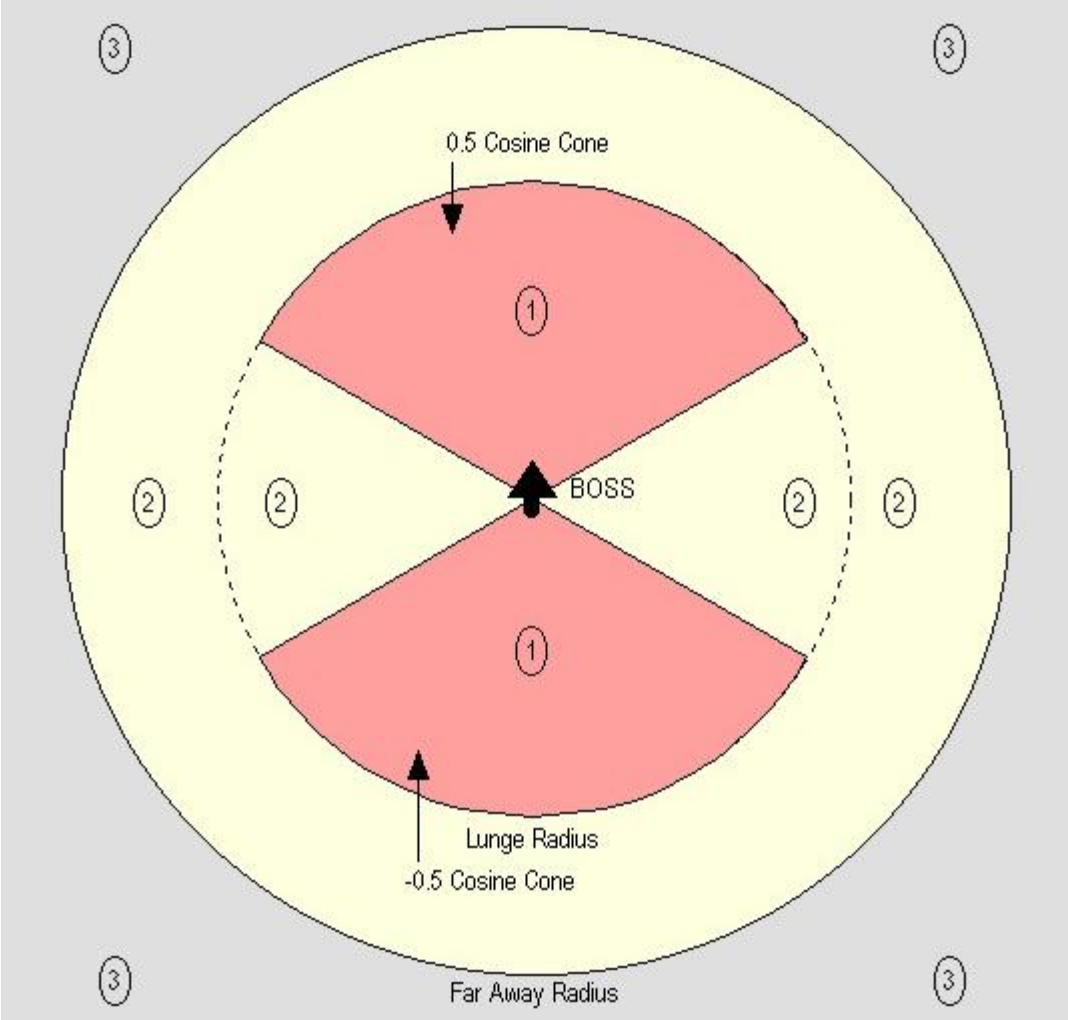
The "too far away" distance is set as follows:

- **`Devil7_FarAway`** (floating point; >= 0.0; default is 1400) specifies the distance (in meter) beyond which the boss judges the player is too far away.

Example:

AIDataValue ("Devil7_FarAway", Float, (Val = 1750.0))

4.2.9.7 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the lunge radius and within either the 0.5 or -0.5 cosine cone of the boss, the boss randomly chooses:	
		Cloak (33.5% chance).
		Clumsy charge (33% chance).
		Circle the player (22.11% chance).
		Zigzag toward the player (11.39% chance).
2	Yellow Zone: If the player is within the far away radius but is outside the red zones, the boss chooses:	
		Cloak (50% chance).
		Chase the player (50% chance).
3	Gray Zone: If the player is outside the too far radius, the boss randomly chooses:	
		Clumsy charge (50% chance).
		Move toward the player (50% chance).

4.2.9.8 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.9.9 Chasing

`AIDataValue ( "ChaseNumTicks", Integer,  ( Val = <#> ))`

The boss chases the player if the player is too far away. This is controlled by:

- **`ChaseNumTicks`** (integer; >= 0; default is 100) specifies the number of ticks of the game clock that the boss chases the player in a single Chase Attack behavior. (1 tick = 0.05 seconds; 1 second = 20 ticks)

Example:

`AIDataValue ( "ChaseNumTicks", Integer,  ( Val = 120 ))`

4.2.9.10 Clumsy Charging

The boss can lunge at the player, using the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`).

4.2.9.11 Circling

The boss can attempt to circle the player, using the Circling the Player or a Location common boss behavior (`circleDistance`).

4.2.9.12 Zigzagging

The boss can zigzag to approach the player. The boss has its effort level set to 2.0 while zigzagging. Zigzagging is hardcoded.

4.2.9.13 Cloaked Attack

The boss can cloak itself while it approaches the player. The boss turns off its cloak to fire its gun and turns the cloak back on after firing.

4.2.10 EXCESSION_10 “Scarab 1”

- *Klown Code File:* exc10.fsm
- *Klown:* DEvil_10
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, TractorBeam2, MoveEngines, OrientEngines, Avoidance, SetTarget, FireControl, Weapons, SearchForTarget, TractorAntiMissile, TargetOffset, Attched, ExcAlive, ExcSound, Skeleton, Scorable, MineControl, SelectiveDamage, Boss10BMineControl
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Has a run away defense against Sinibombs.
- Has tractor beams.
- Launches warriors up to a maximum number; launches more when launched warriors reach or drop below a minimum number.
- Plays a “ChopChop” animation when using its cutting beam on the player.
- If the player is within the boss’s sweep distance and cone, the boss wiggles its arm and activates its cutting beams to try to slice the player.
- Has a set of behaviors based on where the player is in relation to the boss (see diagram in Other Player-Position Behaviors). Overall, the boss will do one of: move toward the player, make a simple attack on the player, lunge at the player, tractor the player, turn toward the player, make a banzai attack, make a clumsy charge on the player, or run away from the player.

4.2.10.1 Scarab Version

```
AIDataValue ( "ScarabVersion",   Integer,   ( Val = <#> ))
```

There are two versions of this boss, `EXCESSION_10` (Scarab 1) and `EXCESSION_10B` (Scarab 2). The following parameter allows the underlying code to distinguish between:

- **`ScarabVersion`** (integer; default is 1) sets the Scarab version; `Val = 1` is `EXCESSION_10` and `Val = 2` is `EXCESSION_10B`.

Note: Since the default is for `EXCESSION_10`, this parameter does not need to be specified for `EXCESSION_10`.

Example:

```
AIDataValue ( "ScarabVersion",   Integer,   ( Val = 1 ))
```

4.2.10.2 Selective Damage

The boss has a selective damage point and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`) and is currently set to take double damage there.

4.2.10.3 Joint Resistance

```
AIDataValue ( "jointResistance",   Float,   ( Val = <#> ))
```

To set the resistance of the joints in the boss's skeleton, use:

- **jointResistance** (floating point; default is 200.0) sets the resistance of the joints in the skeleton; higher numbers make the joints stiffer. If you tune the speed of the boss, you should increase the resistance if you raise the speed and lower the resistance if you drop the speed.

Note: This works the same as `jointResistance` in Common Articulation Parameters, but has its own default.

Example:

```
AIDataValue ( "jointResistance",   Float,   ( Val = 200.0 ))
```

4.2.10.4 Breathing

```
AIDataValue ( "Breath",        Resource, ( Val = ( rtSequence, <resource ID> )))
```

This boss breathes, and the sequence it plays on its joints while breathing is controlled by:

- **Breath** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss breathes.

Example:

```
AIDataValue ( "Breath",        Resource, ( Val = ( rtSequence, EXC10_SEQ_MORPH_BREATH )))

defres rtSequence ( EXC10_SEQ_MORPH_BREATH )
(
   Actions[] = ( ActionId = ( rtMeshMorphAction, EXC10_ACT_MORPH_BREATH ) Time = 0 )
)

defres rtMeshMorphAction ( EXC10_ACT_MORPH_BREATH )
(
   ModelFile     = "monster.dat"
   HierarchyName = "morph_joint"
   AnimType      = kemLoop
   Array[]       = ( ModelName = "boss8_breath01"  Duration = 0.5 )
   Array[]       = ( ModelName = "boss8_breath02"  Duration = 0.5 )
   Array[]       = ( ModelName = "boss8_breath03"  Duration = 0.5 )
   Array[]       = ( ModelName = "boss8_breath02"  Duration = 0.5 )
   Array[]       = ( ModelName = "boss8_breath01"  Duration = 0.5 )
)
```

4.2.10.5 Weaponry

The boss has one cutting-laser gun and two turrets each armed with a cutting laser.

4.2.10.6 Anti-Sinibomb Defenses

```
AIDataValue ( "fleeMissileDistanceMin",   Float,   ( Val = <#> ))
AIDataValue ( "fleeMissileDistanceMax",   Float,   ( Val = <#> ))
```

If the boss detects a Sinibomb that is in front hemisphere of the boss (in the cone defined by the 0.0 cosine angle) and that is close but not too close, it runs away from the bomb, using the Running Away common boss behavior

(runAwayMinAngle, runAwayMaxAngle, runAwayRelVelDevAngle). The boss-specific parameters for this are:

- **fleeMissileDistanceMin** (floating point; default is 600.0) and **fleeMissileDistanceMax** (floating point; default is 1000.0) specify the range of distances (in meters) in which the boss runs away from Sinibombs. The boss does not run away if the Sinibomb is within FleeMissileDistanceMin (as the Sinibomb is judged to close to escape) or is beyond FleeMissileDistanceMax (as the Sinibomb is judged too far away to be a threat).

Example:

```
AIDataValue ( "fleeMissileDistanceMin",   Float,   ( Val =  600.0 ))
AIDataValue ( "fleeMissileDistanceMax",   Float,   ( Val = 1000.0 ))
```

4.2.10.7 Launching Warriors

- *Klown Code File:* exc10.fsm
- *Klown:* Boss10BMineControl
  —*Type:* Submachine

The boss uses the gun XW10_TICK_LAUNCHER to launch warriors. (The gun is currently set to launch five TICK1 strike warriors at a time.) The boss uses some Mine Laying common boss behavior (maxMines, minMines, mineCategory) to control the number of warriors it can have in play at the same time:

- **maxMines** specifies the maximum total number of warriors (not mines, despite the variable's name) launched by the boss that may be in play. (Only warriors "in play" count—a warrior that is destroyed after being launched no longer counts.) The boss launches warriors until the maxMines limit is reached, whereupon it stops launching warriors (but see the next parameter below).
- **minMines** specifies the minimum total number of warriors (not mines) that must be in play for the boss to resume launching warriors once it has stopped launching them. For example, maxMines with Val = 10 and minMines with Val = 5 mean that the boss launches warriors until five are in play. It then stops launching warriors until there are only three in play, whereupon it resumes launching warriors until at least five are again in play. (Note that since warriors are currently launched five at time, maxMines and minMines work best in multiples of five. Tweak these values if you change the number of warriors that are launched at one time.)
- **mineCategory** specifies the resource ID of the weapon category for the boss's warrior launching (not mine laying) weapon. (To launch warriors , the boss must have a warrior launching weapon attached, and this resource ID indicates its weapon category.)

Example:

```
AIDataValue ( "maxMines",     Integer,  ( Val = 10 ))
AIDataValue ( "minMines",     Integer,  ( Val =  5 ))
AIDataValue ( "mineCategory", Integer,  ( Val = GUNCAT_TICKS ))

defres rtGunList( EXCESSION_10 )
(
   Guns[] = ( Category = GUNCAT_TICKS   StartWith = XW10_TICK_LAUNCHER   Enabled = 1 )
)
```

4.2.10.8 Player Too Far

```
AIDataValue ( "tooFar",      Float,    ( Val = <#> ))
```

Use this parameter as follows:

- **tooFar** (floating point; default is 1400.0) specifies the distance (in meters) beyond which the player is judged to be too far away, causing the boss to move toward the player.

Example:

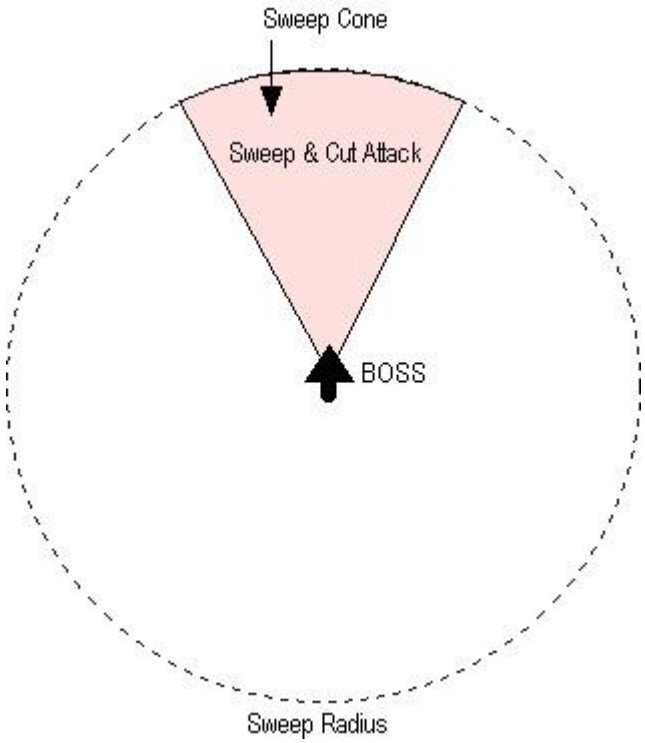
```
AIDataValue ( "tooFar",      Float,    ( Val = 1400.0 ))
```

4.2.10.9 Sweep & Cut Behavior

```
AIDataValue ( "SweepDistance",        Float,    ( Val = <#> ))
AIDataValue ( "SweepCosangle",        Float,    ( Val = <#> ))
AIDataValue ( "CuttingBeamOpen",      Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "CuttingBeamClose",     Resource, ( Val = ( rtSequence, <resource ID> )))
```

If the player is within the boss's sweep distance and in the cone formed by the sweep angle, the boss's Sweep & Cut behavior is triggered. Its "ChopChop" sequence plays, and it uses its cutting beam to attack the player. Use the following parameters:

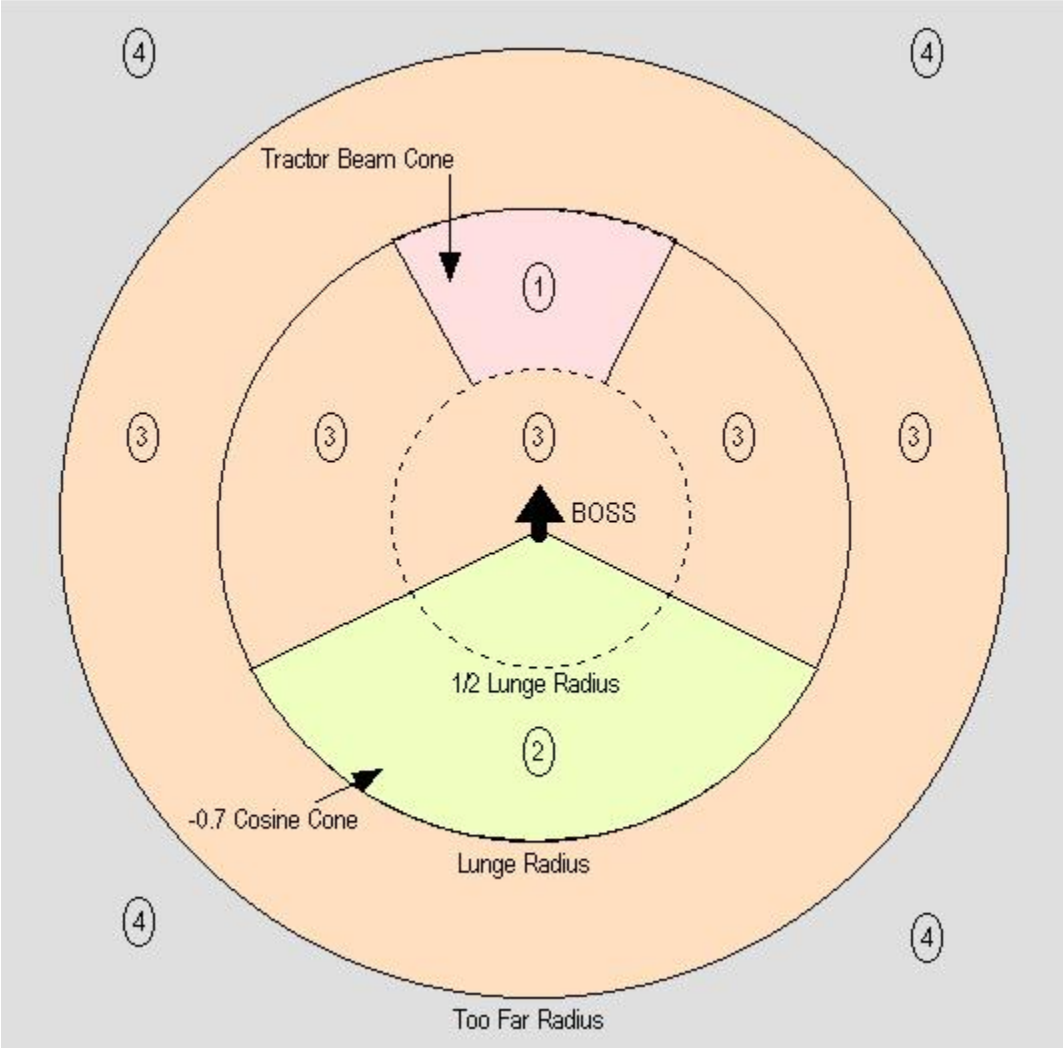
- **SweepDistance** specifies the sweep distance (in meters), within which the boss makes a sweep and cut attack on the player (if the player is also inside the sweep cone).
- **SweepCosangle** specifies the cosine of the angle (see Quick Table of Cosines) used to determine the sweep cone.
- **CuttingBeamOpen** specifies the `rtSequence` resource that plays when the boss uses its cutting beam.
- **CuttingBeamClose** specifies the `rtSequence` resource that plays when the boss ends using its cutting beam.



Example:

```
AIDataValue ( "SweepDistance",    Float,    ( Val = 1000.0  ))
AIDataValue ( "SweepCosangle",    Float,    ( Val =    0.75 ))
AIDataValue ( "CuttingBeamOpen",  Resource, ( Val = ( rtSequence,
                  EXC10_SEQ_CUTTINGBEAMOPEN )))
AIDataValue ( "CuttingBeamClose", Resource, ( Val = ( rtSequence,
                  EXC10_SEQ_CUTTINGBEAMCLOSE )))
```

4.2.10.10 Other Player-Position Behaviors



The position of the player triggers various behaviors of the boss:

1	Pink Zone: If the player is outside half the lunge radius and is within the lunge radius and the tractor beam cone of the boss, the boss randomly chooses:	
		Tractor beam attack (50% chance).
		Simple Attack (50% chance).
2	Yellow Zone: If the player is within the lunge radius and the -0.7 cosine cone (i.e., behind the boss), the boss randomly chooses:	
		Turn toward the player (50% chance).
		Run away (50% chance).

3	Orange Zone: If the player is within the too far radius but is outside both the pink zone and the yellow zone, the boss randomly chooses:	
		Banzai attack (20% chance).
		Run away (35% chance).
		Clumsy charge (15% chance).
		Lunge at the player (30% chance).
4	Gray Zone: If the player is outside the too far radius, the boss moves toward the player (100% chance).	

4.2.10.11 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.10.12 Simple Attack

When the boss makes a simple attack, it fires its turrets on the player.

4.2.10.13 Lunging

The boss can lunge at the player, using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2.10.14 Clumsy Charge

The boss can make at clumsy charge at the player, using the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`).

4.2.10.15 Banzai Attack

The boss lunges at the player using a modified version of the Lunging common boss behavior (including using `lungeDistance`, `lungeAngle`).

4.2.10.16 Tractor Beam Attack

```
AIDataValue ( "TractorBeamCosangle", Float,    ( Val = <#> ))
AIDataValue ( "TractorBeam",         Resource, ( Val = ( rtSequence, <resource ID> )))
```

If the player is in the boss's tractor beam cone, the boss can activate the tractor beam, pulls the player to a position, and holds the player there until the tractor beam attack ends. (While caught in the beam, the player can be attacked by other weapons and other means, such as by the warriors launched by the boss. The tractor beam uses the Tractor Beam 2 parameters for its tractor beam (`tractorBeam` and up to 21 more parameters), plus these additional parameters:

- **`TractorBeamCosangle`** (floating point; default is 0.75) specifies the cosine of the angle (see Quick Table of Cosines) used to determine the tractor beam cone.
- **`TractorBeam`** (resource; no default) specifies the `rtSequence` resource that plays when the boss uses its tractor beam.

Example:

```
AIDataValue ( "TractorBeamCosangle", Float,    ( Val = 0.65 ))
AIDataValue ( "TractorBeam",      Resource, ( Val = ( rtSequence, EXC10_SEQ_TRACTORBEAM
                ))))
```

4.2.10.17 Running Away

The boss can run away from the player or Sinibombs, using the Running Away common boss behavior (`runAwayMinAngle`, `runAwayMaxAngle`, `runAwayRelVelDevAngle`).

4.2.11 EXCESSION_10B ("Scarab 2")

- *Klown Code File:* exc10.fsm
- *Klown:* DEvil_10
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, TractorBeam2, MoveEngines, OrientEngines, Avoidance, SetTarget, FireControl, Weapons, SearchForTarget, TractorAntiMissile, TargetOffset, Attched, ExcAlive, ExcSound, Skeleton, Scorable, MineControl, SelectiveDamage, Boss10BMineControl
  —*Has Internal Klown:* NormalBehavior

This boss is a modified version of `EXCESSION_10`. The section for `EXCESSION_10` is repeated here, with changes for `EXCESSION_10B` marked in red.

The behaviors for this boss are:

- Has a selective damage point.
- Has a run away defense against Sinibombs.
- Has tractor beams.
- Launches warriors up to a maximum number; launches more when launched warriors reach or drop below a minimum number.
- Plays a "ChopChop" animation when using its cutting beam on the player.
- If the player is within the boss's sweep distance and cone, the boss wiggles its arm and activates its cutting beams to try to slice the player.
- Has a set of behaviors based on where the player is in relation to the boss (see diagram in Other Player-Position Behaviors). Overall, the boss will do one of: move toward the player, make a simple attack on the player, lunge at the player, tractor the player, turn toward the player, lay mines (instead of make a banzai attack), or move away from the player.

4.2.11.1 Scarab Version

```
AIDataValue ( "ScarabVersion",   Integer,   ( Val = <#> ))
```

There are two versions of this boss, `EXCESSION_10` (Scarab 1) and `EXCESSION_10B` (Scarab 2). The following parameter allows the underlying code to distinguish between:

- **`ScarabVersion`** (integer; default is 1) sets the Scarab version; `Val = 1` is `EXCESSION_10` and `Val = 2` is `EXCESSION_10B`.

Note: Since the default is for `EXCESSION_10`, this parameter must be specified for `EXCESSION_10B`.

Example:

```
AIDataValue ( "ScarabVersion",   Integer,   ( Val = 1 ))
```

4.2.11.2 Selective Damage

The boss has a selective damage point and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`) and is currently set to take double damage there.

4.2.11.3 Joint Resistance

```
AIDataValue ( "jointResistance",   Float,   ( Val = <#> ))
```

To set the resistance of the joints in the boss's skeleton, use:

- **jointResistance** (floating point; default is 200.0) sets the resistance of the joints in the skeleton; higher numbers make the joints stiffer. If you tune the speed of the boss, you should increase the resistance if you raise the speed and lower the resistance if you drop the speed.

Note: This works the same as `jointResistance` in Common Articulation Parameters, but has its own default.

Example:

```
AIDataValue ( "jointResistance",   Float,   ( Val = 200.0 ))
```

4.2.11.4 Breathing

```
AIDataValue ( "Breath",        Resource, ( Val = ( rtSequence, <resource ID> )))
```

This boss breathes, and the sequence it plays on its joints while breathing is controlled by:

- **Breath** (resource; no default) specifies the `rtSequence` resource that plays on the spikes when the boss breathes.

Example:

```
AIDataValue ( "Breath",        Resource, ( Val = ( rtSequence, EXC10B_SEQ_MORPH_BREATH )))

defres rtSequence ( EXC10B_SEQ_MORPH_BREATH )
(
   Actions[] = ( ActionId = ( rtMeshMorphAction, EXC10B_ACT_MORPH_BREATH ) Time = 0 )
)

defres rtMeshMorphAction ( EXC10B_ACT_MORPH_BREATH )
(
   ModelFile     = "monster.dat"
   HierarchyName = "morph_joint"
   AnimType      = kemLoop
   Array[]       = ( ModelName = "boss8b_breath01" Duration = 0.5 )
   Array[]       = ( ModelName = "boss8b_breath02" Duration = 0.5 )
   Array[]       = ( ModelName = "boss8b_breath01" Duration = 0.5 )
)
```

4.2.11.5 Wing Flapping

```
AIDataValue ( "OpenWings",        Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "CloseWings",       Resource, ( Val = ( rtSequence, <resource ID> )))
```

This boss opens its wings when it is going to lay mines and closes them when it finishes laying mines:

- **OpenWings** (resource; no default) specifies the `rtSequence` resource that plays when the boss opens it wings.
- **CloseWings** (resource; no default) specifies the `rtSequence` resource that plays when the boss opens it wings.

Example:

```
AIDataValue ( "OpenWings",        Resource, ( Val = ( rtSequence, EXC10B_SEQ_OPENWINGS )))
AIDataValue ( "CloseWings",       Resource, ( Val = ( rtSequence, EXC10B_SEQ_CLOSEWINGS
                  )))
```

4.2.11.6 Weaponry

The boss has one cutting-laser gun, ~~and~~ two turrets each armed with a cutting laser, and ten turrets that launch mines.

4.2.11.7 Anti-Sinibomb Defenses

```
AIDataValue ( "fleeMissileDistanceMin",   Float,   ( Val = <#> ))
AIDataValue ( "fleeMissileDistanceMax",   Float,   ( Val = <#> ))
```

If the boss detects a Sinibomb that is in front hemisphere of the boss (in the cone defined by the 0.0 cosine angle) and that is close but not too close, it runs away from the bomb, using the Running Away common boss behavior (`runAwayMinAngle`, `runAwayMaxAngle`, `runAwayRelVelDevAngle`). The boss-specific parameters for this are:

- **`fleeMissileDistanceMin`** (floating point; default is 600.0) and **`fleeMissileDistanceMax`** (floating point; default is 1000.0) specify the range of distances (in meters) in which the boss runs away from Sinibombs. The boss does not run away if the Sinibomb is within `FleeMissileDistanceMin` (as the Sinibomb is judged to close to escape) or is beyond `FleeMissileDistanceMax` (as the Sinibomb is judged too far away to be a threat).

Example:

```
AIDataValue ( "fleeMissileDistanceMin",   Float,   ( Val =  600.0 ))
AIDataValue ( "fleeMissileDistanceMax",   Float,   ( Val = 1000.0 ))
```

4.2.11.8 Launching Warriors

- *Klown Code File:* exc10.fsm
- *Klown:* Boss10BMineConrol
  —*Type:* Submachine

The boss uses the gun `XW10_TICK_LAUNCHER` to launch warriors. (The gun is currently set to launch five `TICK1` strike warriors at a time.) The boss uses some Mine Laying common boss behavior (`maxMines`, `minMines`, `mineCategory`) to control the number of warriors it can have in play at the same time:

- **`maxMines`** specifies the maximum total number of warriors (not mines, despite the variable's name) launched by the boss that may be in play. (Only warriors "in play" count—a warrior that is destroyed after being launched no longer counts.) The boss launches warriors until the `maxMines` limit is reached, whereupon it stops launching warriors (but see the next parameter below).

- **`minMines`** specifies the minimum total number of warriors (not mines) that must be in play for the boss to resume launching warriors once it has stopped launching them. For example, `maxMines` with `Val = 10` and `minMines` with `Val = 5` mean that the boss launches warriors until five are in play. It then stops launching warriors until there are only three in play, whereupon it resumes launching warriors until at least five are again in play. (Note that since warriors are currently launched five at time, `maxMines` and `minMines` work best in multiples of five. Tweak these values if you change the number of warriors that are launched at one time.)

- **mineCategory** specifies the resource ID of the weapon category for the boss's warrior launching (not mine laying) weapon. (To launch warriors , the boss must have a warrior launching weapon attached, and this resource ID indicates its weapon category.)

Example:

```
AIDataValue ( "maxMines",     Integer,  ( Val = 10 ))
AIDataValue ( "minMines",     Integer,  ( Val =  5 ))
AIDataValue ( "mineCategory", Integer,  ( Val = GUNCAT_TICKS ))

defres rtGunList( EXCESSION_10 )
(
   Guns[] = ( Category = GUNCAT_TICKS   StartWith = XW10_TICK_LAUNCHER   Enabled = 1 )
)
```

4.2.11.9 Player Too Far

```
AIDataValue ( "tooFar",      Float,    ( Val = <#> ))
```

Use this parameter as follows:

- **tooFar** (floating point; default is 1400.0) specifies the distance (in meters) beyond which the player is judged to be too far away, causing the boss to move toward the player.

Example:

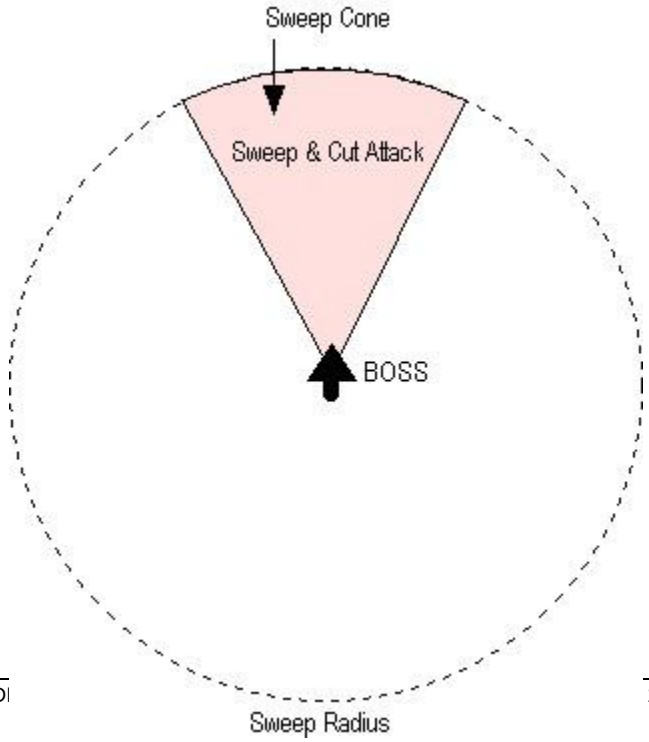
```
AIDataValue ( "tooFar",      Float,    ( Val = 1400.0 ))
```

4.2.11.10 Sweep & Cut Behavior

```
AIDataValue ( "SweepDistance",        Float,    ( Val = <#> ))
AIDataValue ( "SweepCosangle",        Float,    ( Val = <#> ))
AIDataValue ( "CuttingBeamOpen",      Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "CuttingBeamClose",     Resource, ( Val = ( rtSequence, <resource ID> )))
```

If the player is within the boss's sweep distance and in the cone formed by the sweep angle, the boss's Sweep & Cut behavior is triggered. Its "ChopChop" sequence plays, and it uses its cutting beam to attack the player. Use the following parameters:

- **SweepDistance** specifies the sweep distance (in meters), within which the boss makes a sweep and cut attack on the player (if the player is also inside the sweep cone).
- **SweepCosangle** specifies the cosine of the angle (see Quick Table of Cosines) used to determine the sweep cone.
- **CuttingBeamOpen** specifies the `rtSequence` resource that plays when the boss uses its cutting beam.

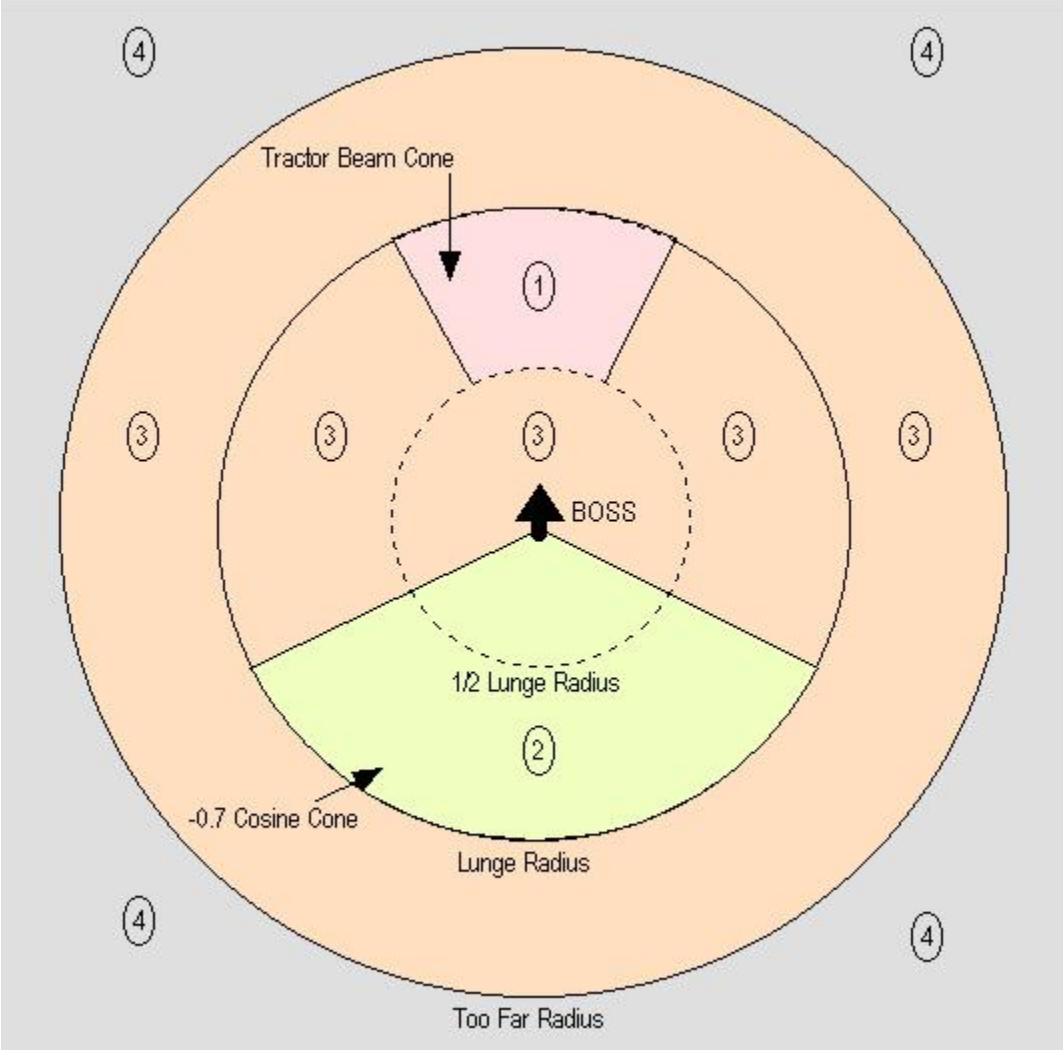


- **`CuttingBeamClose`** specifies the `rtSequence` resource that plays when the boss ends using its cutting beam.

Example:

```
AIDataValue ( "SweepDistance",    Float,    ( Val = 1000.0  ))
AIDataValue ( "SweepCosangle",    Float,    ( Val =    0.75 ))
AIDataValue ( "CuttingBeamOpen",  Resource, ( Val = ( rtSequence,
                EXC10_SEQ_CUTTINGBEAMOPEN )))
AIDataValue ( "CuttingBeamClose", Resource, ( Val = ( rtSequence,
                EXC10_SEQ_CUTTINGBEAMCLOSE )))
```

4.2.11.11 Other Player-Position Behaviors



The position of the player triggers various behaviors of the boss:

1	Pink Zone: If the player is outside half the lunge radius and is within the lunge radius and the tractor beam cone of the boss, the boss randomly chooses:	
		Tractor beam attack (50% chance).

		Simple Attack (50% chance).
2	Yellow Zone: If the player is within the lunge radius and the -0.7 cosine cone (i.e., behind the boss), the boss randomly chooses:	
		Turn toward the player (50% chance).
		Run away (50% chance).
3	Orange Zone: If the player is within the too far radius but is outside both the pink zone and the yellow zone, the boss randomly chooses:	
		Mine attack (40% chance).
		Run away (40% chance).
		~~Clumsy charge (15% chance).~~
		Lunge at the player (20% chance).
4	Gray Zone: If the player is outside the too far radius, the boss moves toward the player (100% chance).	

4.2.11.12 Simple Attack

When the boss makes a simple attack, it fires its turrets on the player.

4.2.11.13 Lunging

The boss can lunge at the player, using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

~~4.2.11.14 Clumsy Charge~~

~~The boss can make at clumsy charge at the player, using the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`).~~

4.2.11.15 Mine Laying

(This boss has mine laying in place of Banzai Attack.)

```
defres rtAI ( EXC10B_TURRET_MINE )
(
    ...
AIDataValue ( "maxShotFired",       Integer,   ( Val = <#> ))
    ...
)
```

The boss can lay mines, using the Mine Laying common boss behavior (`maxMines`, `minMines`, `mineCategory`). The boss has ten invisible mine laying turrets on its back, which are controlled (in turret.r) by the `EXC10B_TURRET_MINE` resources. Their mine laying is controlled by the following parameter (note that this is in the turret's AI resource, not the boss's AI resource):

- **`maxShotFired`** specifies the number of mines that a mine laying turret weapon may be in play. (Only mines "in play" count—a mine that is destroyed after being laid no longer counts.) The weapon lays mines until the `maxShotFired` limit is reached, whereupon it stops laying mines until it is below its limit again.

Note: The weapon's AI resource also use `fireConeDistance` and `fireConeAngle` parameters, which the ChargingTurret klown. These work identical to the same-named parameters in the FireControl klown. However, unlike the FireControl klown, the ChargingTurret klown does not use a `fireConeNormal` parameter.

Example:

```
defres rtAI ( EXC10B_TURRET_MINE )
(
   ...
AIDataValue ( "maxShotFired",      Integer,  ( Val = 5 ))
   ...
)
```

4.2.11.16 Tractor Beam Attack

```
AIDataValue ( "TractorBeamCosangle", Float,    ( Val = <#> ))
AIDataValue ( "TractorBeam",         Resource, ( Val = ( rtSequence, <resource ID> )))
```

If the player is in the boss's tractor beam cone, the boss can activate the tractor beam, pulls the player to a position, and holds the player there until the tractor beam attack ends. (While caught in the beam, the player can be attacked by other weapons and other means, such as by the warriors launched by the boss. The tractor beam uses the Tractor Beam 2 parameters for its tractor beam (`tractorBeam` and up to 21 more parameters), plus these additional parameters:

- **`TractorBeamCosangle`** specifies the cosine of the angle (see Quick Table of Cosines) used to determine the tractor beam cone.
- **`TractorBeam`** specifies the `rtSequence` resource that plays when the boss uses its tractor beam.

Example:

```
AIDataValue ( "TractorBeamCosangle", Float,    ( Val = 0.65 ))
AIDataValue ( "TractorBeam",      Resource, ( Val = ( rtSequence, EXC10_SEQ_TRACTORBEAM
                 )))
```

4.2.11.17 Running Away

The boss can run away from the player or Sinibombs, using the Running Away common boss behavior (`runAwayMinAngle`, `runAwayMaxAngle`, `runAwayRelVelDevAngle`).

4.2.12 EXCESSION_11 “Bull”

- *Klown Code File:* exc11.fsm
- *Klown:* DEvil_11
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, Avoidance, SetTarget, FireControl, Weapons, AntiMissileDefense, SearchForTarget, TargetOffset, ExcAlive, ExcSound, Attached
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- It has a chewing articulation sequence.
- It has two tail articulation sequences.
- It can teleport, to close on the player.
- It tries to run away from Sinibombs.
- It can lunge at and stay close to the player.
- Otherwise it circles the player.

4.2.12.1 Selective Damage

The boss has a selective damage point on its belly and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.12.2 Chewing

```
AIDataValue ( "Chew",   Resource,   ( Val = ( rtSequence, <resource ID> )))
```

The following parameter sets the boss’s chewing sequence:

- **`Chew`** (resource; no default) specifies the `rtSequence` resource that plays when the boss chews its mandibles.

Example:

```
AIDataValue ( "Chew",   Resource,   ( Val = ( rtSequence, EXC11_SEQ_ANIM_CHEW )))
```

4.2.12.3 Tail Articulations

```
AIDataValue ( "Sway",     Resource,   ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "Unroll",   Resource,   ( Val = ( rtSequence, <resource ID> )))
```

The following parameters set the boss’s tail articulation sequences:

- **`Sway`** (resource; no default) specifies the `rtSequence` resource that plays when the boss sways its tail.
- **`Unroll`** (resource; no default) specifies the `rtSequence` resource that plays when the boss unrolls its tail.

Example:

```
AIDataValue ( "Sway",     Resource,   ( Val = ( rtSequence, EXC11_SEQ_MORPH_SWAY   )))
AIDataValue ( "Unroll",   Resource,   ( Val = ( rtSequence, EXC11_SEQ_MORPH_UNROLL )))
```

4.2.12.4 Weaponry

The boss has an energy gun that emits green plasma shots and four missile-firing turrets.

4.2.12.5 Anti-Sinibomb Defenses

```
AIDataValue ( "TeleportDistanceMissile",   Float,   ( Val = <#> )
AIDataValue ( "RunAwayDistanceGoal",       Float,   ( Val = <#> )
```

The boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `antiMissileCosAngle`, `antiMissileCategory`) to detect incoming Sinibombs. It tries to flee Sinibombs, using the Fleeing common boss behavior (`fleeDistance`). It has its effort level set to 2.5 while fleeing. The following parameters further control the boss's run away defense:

- **`TeleportDistanceMissile`** (floating point; default it 400.0) specifies the distance (in meters) beyond which, if the boss detects an incoming Sinibomb, it runs away from the Sinibomb. (Sinibombs closer than `TeleportDistanceMissile` are assumed to be too close to run away from.)
- **`RunAwayDistanceGoal`** (floating point; default it 1500.0) specifies the distance (in meters) from the player the boss runs away to when fleeing a Sinibomb. (When it decides to run away from a Sinibomb, it picks a point `RunAwayDistanceGoal` from the player's current position and moves there.)

Example:

```
AIDataValue ( "TeleportDistanceMissile",   Float,   ( Val =  400.0 ))
AIDataValue ( "RunAwayDistanceGoal",       Float,   ( Val = 1500.0 ))
```

4.2.12.6 Behaviors Triggered by Player Position

The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the teleportation activation radius, beyond half the lunge radius, and outside the -0.5 cosine cone, the boss randomly chooses:	
		Move toward the player (50% chance).
		Lunge at the player (50% chance).
2	Yellow Zone: If the player is within the teleportation activation radius, beyond half the lunge radius, and inside the -0.5 cosine cone, the boss randomly chooses:	
		To teleport (25% chance).
		To circle the player (75% chance).
3	Gray Zone: If the player is outside the teleportation activation radius, the boss teleports (100% chance).	

4.2.12.7 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.12.8 Teleportation

```
AIDataValue ( "TeleportDistance", Float,    ( Val = <#> ))
```

The boss uses the common Teleportation parameters (`FadeOut`, `FadeIn`, `TeleportDistanceToTarget`, `TeleportMaxEnergy`, `TeleportMinEnergy`, and `DamageLevelToAbort`), plus the following parameter:

- **`TeleportDistance`** (floating point; default it 700.0) specifies the distance (in meters) that the boss reappears in front of the player when it teleports. (*Note:* This parameter and `TeleportDistanceToTarget` should be set to the same value.)

Example:

```
AIDataValue ( "TeleportDistance",          Float,    ( Val = 700.0 ))
AIDataValue ( "FadeOut",                   Resource, ( Val = ( rtSequence,
                  EXC11_SEQ_TELEPORT_FADEOUT )))
AIDataValue ( "FadeIn",                    Resource, ( Val = ( rtSequence,
                  EXC11_SEQ_TELEPORT_FADEIN )))
AIDataValue ( "TeleportDistanceToTarget", Float,    ( Val = 700.0 ))
AIDataValue ( "TeleportMaxEnergy",         Float,    ( Val = 100.0 ))
AIDataValue ( "DamageLevelToAbort",        Float,    ( Val =  10.0 ))
AIDataValue ( "TeleportMinEnergy",         Float,    ( Val =  10.0 ))
```

4.2.12.9 Lunging

The boss can lunge at the player, using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2.12.10 Circling

The boss can circle the player, using the Circling the Player or a Location common boss behavior (`circleDistance`).

4.2.13 No EXCESSION_12

Level 12 does not have a boss.

4.2.14 EXCESSION_13 “Mother Ship”

- *Klown Code File:* exc13.fsm
- *Klown:* DEvil_13
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SetTarget, FireControl, Weapons, SearchForTarget, ExcAlive, ExcSound, Attached, LaunchMother, Skeleton, SelectiveDamage, TargetOffset, Scorable, AntiMissileDetector
  —*Has Internal Klown:* NormalBehavior

The behaviors for this boss are:

- Initially, moves out 1,000 meters from the gate.
- Every so often, launches two warriors off its back.
- Does a cutting attack, where it runs away, taunts with “Run, coward!”, then charges at the player while zapping its cutting lasers.
- Does a spin attack where it points its back turrets, then its bottom turrets, then its front beam turrets at the player. During this time, it ignores incoming Sinibombs.
- Does a circling attack, where it circles the player and launches a bevy of slow missiles.
- After a while, if it is not attacked, it goes away and circles in the distance for a while, giving the player a break.
- If it cannot detect the player (such as if the player is cloaked), the boss parks at the gate.
- If it senses an incoming Sinibomb, it stops what it’s doing and either runs or tries to turn its hard-shelled back toward the Sinibomb.

4.2.14.1 Selective Damage

The boss has a selective damage point on its big fat green belly and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.14.2 Weaponry

The boss currently has the following weaponry: 1) a turret on its left claw that mounts a burster, 2) a turret on its right claw that apparently doesn’t yet have a weapon mounted, 3) a laser-armed turret, 4) a missile turret, 5) a missile tube that launches “slow but deadly” missiles (three at a time), and 6) a cutting beam gun.

The turret on the boss’s left claw uses the Turret klown and has its own `rtAI` resource (which uses the standard AI parameters for turrets). See `defres rtAI ( ESC13_LEFTCLAW_TURRET )` in monster.r for further details.

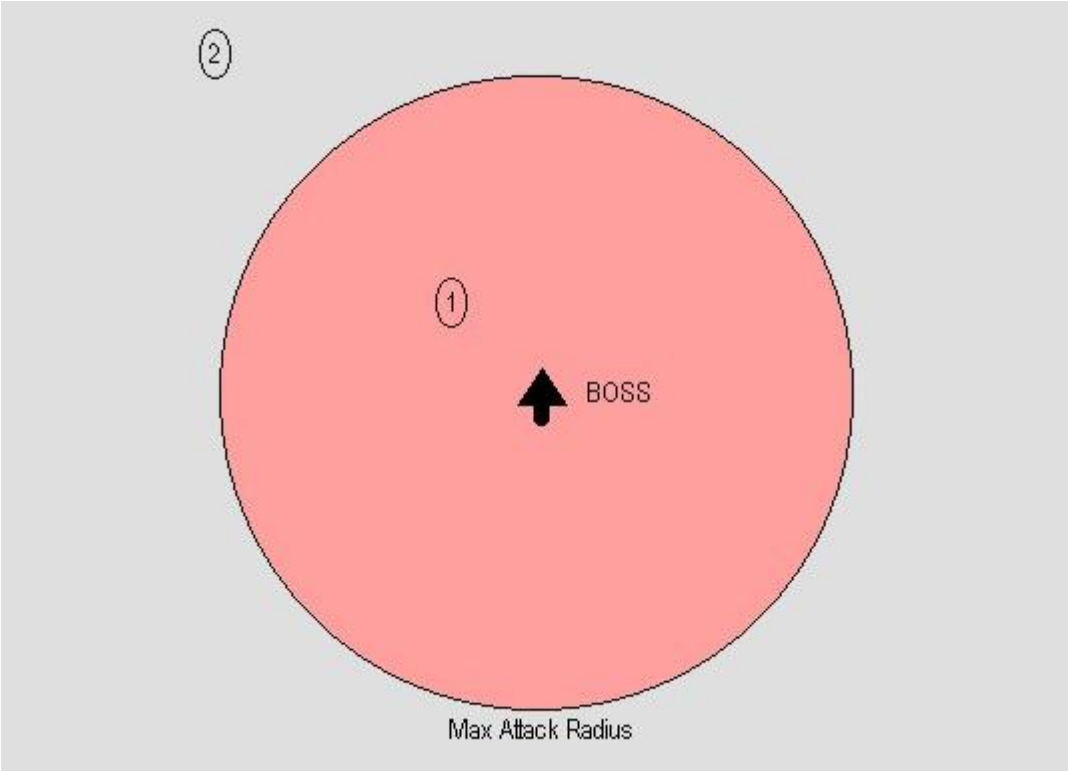
4.2.14.3 Anti-Sinibomb Defenses

This boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `onlyDetectSinibombs`) to trigger its defense against incoming Sinibombs:

50% chance the boss turns its armored back towards the Sinibomb. While turning its back to a Sinibomb, the boss has its effort level set to 2.0.

50% chance the boss runs away.

4.2.14.4 Behaviors Triggered by Player Position



Out of every 16 attacks the boss makes, one attack consists of the boss launching warriors. For the other attacks, the position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the boss's maximum attack radius, the boss randomly chooses:	
		To make a spin turret attack (33.33% to 0% chance, decreasing in intervals of 10%). But see note!
		To make a cutting attack (33.33% to 0% chance, decreasing in intervals of 10%). But see note!
		To make a circling attack (33.33% to 0% chance, decreasing in intervals of 10%). But see note!
		To break off and flee (0.0% to 100% chance, increasing in intervals of 10%). But see note!

	Note: The code for the boss deciding to break off and flee has been modified so that the boss will never decide to do this (and thus the first three actions above always each have a 33.33% chance of occurring. The code to bias the boss to flee still exists, and if it is turned back on, it will work as follows: The specific probability of one of the above actions occurring varies. Initially, the boss has a 10% chance of breaking off action and running away. The probability for this break off action increases by 10% overall each time the boss does a red-zone action. However, the probability is reset to 10% if either the boss does break off action or if the boss detects an incoming Sinibomb.

2	Gray Zone: If the player is outside the boss's maximum attack radius, the boss randomly chooses:	
		To circle (100% to 0% chance, decreasing in intervals of 30%). But see note!
		To move toward the player (0% to 100% chance, increasing in intervals of 30%). But see note!
	Note: The code for the boss deciding to move toward the player has been modified so that the boss will never decide to do this (and thus the first action above always has a 100% chance of occurring. The code to bias the boss to move toward the player still exists, and if it is turned back on, it will work as follows: The specific probability of one of the above actions occurring varies. Initially, the boss has a 0% chance of moving toward the player. The probability for this action increases by 30% overall each time the boss decides to circle in the gray zone.	

4.2.14.5 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 1,000 meters away before doing anything else.

4.2.14.6 Parking at the Gate

If the boss cannot detect the player (such as if the player is cloaked), the boss parks at the gate using the Park at Gate common boss behavior.

4.2.14.7 Running Away

The boss employs the Running Away common boss behavior to run away from Sinibombs (`runAwayMinAngle`, `runAwayMaxAngle`, `runAwayRelVelDevAngle`). While running away, the boss has its effort level set to 2.0.

4.2.14.8 Taking a Break

`AIDataValue ( "behaviorTime",  Float,  ( Val = <#> ))`

Every so often, if the boss is not being attacked, it takes a union-mandated break for a while and gives the player a chance to do something other than fight:

- **`behaviorTime`** (floating point; default is 7.0) specifies the amount of time (in seconds) that the boss circles its home position (without attacking the player) while it takes a break.

Example:

`AIDataValue ( "behaviorTime",  Float,  ( Val = 7.0 ))`

4.2.14.9 Cutting Attack

This boss first flees using the Fleeing common boss behavior (`fleeDistance`) to get some distance away from its target. It then orients at its enemy and uses the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`) for its "cutting" attack, firing its cutting laser turrets as it charges.

4.2.14.10 Spin Turret Attack

`AIDataValue ( "spinHoldTime", Float,  ( Val = <#> ))`

- **spinHoldTime** (floating point; default is 3.0) specifies the amount of time (in seconds) in which the boss makes a spin turret attack. During the attack, the boss first points its tail, then its belly, then its front toward the player. If the boss is damaged during its spin turret attack, it immediately goes to the end of its attack, pointing its cutting beams at the player.

Example:

```
AIDataValue ( "spinHoldTime", Float,    ( Val = 2.0 ))
```

4.2.14.11 Circling Attack

```
AIDataValue ( "numMissilesPerAttack", Integer,    ( Val = 3 ))
```

- **numMissilesPerAttack** (integer; default is 3) specifies the number of missiles the boss fires when making a circling attack. The boss circles the player (or, in some instances, its home location) using the Circling the Player or a Location common boss behavior (`circleDistance`), and continues to circle until it has fired the specified number of slow missiles. The slow missiles are assumed to be fired by the gun with category `GUNCAT_MINES` (which in this case is `EXW13_SLOW_MISSILE` located in missile.r).

Example:

```
AIDataValue ( "numMissilesPerAttack", Integer,    ( Val = 3 ))
```

4.2.14.12 Launching Fighters

The boss occasionally launches two fighters using the Ship Launching common ship behavior (`launchList`, `launchDistance`). The boss's code currently is hardwired to launch the fighters once every 16 attacks.

4.2.15 EXCESSION_14 “Mine Layer”

- *Klown Code File:* exc14.fsm
- *Klown:* DEvil_14
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, Avoidance, SetTarget, SearchForTarget, Weapons, AntiMissileDefense, TargetOffset, Skeleton, SelectiveDamage, ExcAlive, ExcSound, FireControl, Attached, MineControl
  —*Has Internal Klown:* NormalBehavior

This boss is “a pretty sluggish piece of meat” that relies on its mine laying ability to fight the player. Its behaviors are:

- Moves its arms, as if it was swimming, chews, and breathes.
- Can flee Sinibombs and lay mines while doing so.
- If beyond its `maxAttackDistance` from the player, moves toward the player.
- When within its `maxAttackDistance` from the player, randomly selects between: (30% chance) circling the player and laying mines, or (70% chance) zigzagging and laying mines.
- Has a selective damage point.

4.2.15.1 Selective Damage

The boss has a selective damage point on belly and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.15.2 Articulation Sequences

```
AIDataValue ( "Breath",     Resource, ( Val = ( rtSequence, <resource ID> ) ))
AIDataValue ( "Chew",       Resource, ( Val = ( rtSequence, <resource ID> ) ))
AIDataValue ( "RightWave",  Resource, ( Val = ( rtSequence, <resource ID> ) ))
AIDataValue ( "LeftWave",   Resource, ( Val = ( rtSequence, <resource ID> ) ))
```

The following parameters set the boss’s articulation sequences:

- **`Breath`** (resource; no default) specifies the `rtSequence` resource that plays when the boss breathes.
- **`Chew`** (resource; no default) specifies the `rtSequence` resource that plays when the boss chews with its mandibles.
- **`RightWave`** (resource; no default) specifies the `rtSequence` resource that plays for one of the boss’s swimming articulations.
- **`LeftWave`** (resource; no default) specifies the `rtSequence` resource that plays for the other of the boss’s swimming articulations.

Example:

```
AIDataValue ( "Breath",    Resource, ( Val = ( rtSequence, EXC14_SEQ_MORPH_BREATH     ) ))
AIDataValue ( "RightWave", Resource, ( Val = ( rtSequence, EXC14_SEQ_MORPH_RIGHT_WAVE ) ))
AIDataValue ( "LeftWave",  Resource, ( Val = ( rtSequence, EXC14_SEQ_MORPH_LEFT_WAVE  ) ))
AIDataValue ( "Chew",      Resource, ( Val = ( rtSequence, EXC14_SEQ_ANIM_CHEW        ) ))
```

4.2.15.3 Weaponry

The boss has a mine-laying gun, a beam gun, and a gun that launches four short-ranged missiles at a time.

4.2.15.4 Anti-Sinibomb Defenses

`AIDataValue ( "FleeActivationDistanceFromMissile", Float, ( Val = <#> ))`

The boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `onlyDetectSinibombs`) to detect Sinibombs. The boss flees (using the Fleeing common boss behavior, `fleeDistance`) an incoming Sinibomb and has its effort level set to 2.5 while fleeing. It also uses the following:

- **`FleeActivationDistanceFromMissile`** (floating point; default is 700.0) specifies the distance (in meters) beyond which the boss flees when it detects an incoming Sinibomb. (At distances within `FleeActivationDistanceFromMissile`, the boss judges it cannot escape and does not flee.)

While fleeing, the boss can lay mines.

Example:

`AIDataValue ( "FleeActivationDistanceFromMissile", Float, ( Val = 700.0 ))`

4.2.15.5 Behaviors Triggered by Player Position

The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the boss's maximum attack radius, the boss randomly chooses:	
		To circle and lay mines (30% chance).
		To zigzag and lay mines (70% chance).

2	Gray Zone: If the player is outside the boss's maximum attack radius, the boss moves toward the player (100% chance):

4.2.15.6 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.15.7 Mine Laying

The boss uses the Mine Laying common boss behavior (`maxMines`, `minMines`, `mineCategory`) to lay mines.

4.2.15.8 Circling

The boss uses the Circling the Player or a Location common boss behavior (`circleDistance`) to circle the player. It can lay mines while circling.

4.2.15.9 Zigzagging

The boss can move in zigzags and lay mines. This behavior is not tunable.

4.2.16 EXCESSION_15 “Cluster”

- *Klown Code File:* exc15.fsm
- *Klown:* DEvil_15
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, Skeleton, ExcPain, Attached, SetTarget, FireControl, Weapons, SearchForTarget, LaunchChild, ExcAlive, ExcSound, Scorable
  —*Has Internal Klown:* NormalBehavior

- *Klown Code File:* exc15.fsm
- *Klown:* DEvil_15Core
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SpinEngines, SetTarget, SearchForTarget, LaunchMother, ExcAlive, AntiMissileDetector
  —*Has Internal Klowns:* AngularBehavior, NormalBehavior

The behaviors for this boss are:

- Initially moves out to 800 meters from the gate.
- It is made up of five different parts, each with a different weapon: shots, concussion, lightning bolt, cutting laser, and missiles. Overall, Cluster has three cutting beams, three lightning guns, and six of everything else (missile, concussion, normal shot).
- While joined, it does one of two linear movements: either moving to arbitrary points around the player, or a ramming attack.
- While joined, it orients independently of its movements. It either spins around an arbitrary axis or aims each of its five parts at the player.
- After being hit three times by Sinibombs, it splits into five pieces. It will rejoin after doing some more attacks.
- While split, it does one of two attacks: A piece tries to ram the player, or a piece stops and acts like a turret, firing on the player. The split attacks are coordinated so only one piece is attacking at once, while the other pieces circle the player.

There are two totally different klowns responsible for making Cluster work; one is the “group” klown, the other is the “individual” klown. (Both klowns share the Waving Arms behavior.)

4.2.16.1 Arm Waving

```
AIDataValue ( "armAngle",      Float,    ( Val = <#> ))
AIDataValue ( "armSweepAngle", Float,    ( Val = <#> ))
```

Whether or not Cluster is grouped or in individual pieces, its individual pieces wave their tentacles around, aiming them at the player. Two parameters affect the movement:

- **armAngle** (floating point; default is 35.0) describes the maximum allowable offset angle (in degrees) that each tentacle may move from the “rest” direction.
- **armSweepAngle** (floating point; default is 30.0) describes the size of a “sweep” angle (in degrees) through which the tentacle oscillates as it aims.

Example:

```
AIDataValue ( "armAngle",      Float,    ( Val = 35.0 ))
```

```
AIDataValue ( "armSweepAngle", Float,    ( Val = 10.0 ))
```

4.2.16.2 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.16.3 Behavior Summary

Group	Cluster, when it is grouped, travels like an inertialess UFO. It can move equally well in any direction and doesn't have any well-defined "front." Because of this, its angular motion and its linear motion are totally decoupled. Cluster randomly picks between two spin behaviors and, separately, randomly picks between two linear movement behaviors. After each spin or linear behavior completes, it randomly picks another spin or linear behavior. For example, when one spin behavior ends, it picks another one, regardless of what its linear movement behavior may currently be. Cluster attempts to run away from Sinibombs. When hit by a Sinibomb, Cluster breaks up into its individual pieces.
Individual	Cluster pieces, when not grouped together, are no longer inertialess. Each piece's angular motion and linear motion are coupled, just like all other inertial ships. When Cluster is in this state, it doesn't use any of the group behaviors described above. The individual pieces can take turns aiming and sniping at the player or attempting to ram the player. The split attacks are coordinated so only one piece is attacking at once, while the other pieces circle the player. After every two attacks, the individual pieces attempt to reform into a group again.

4.2.16.4 Group Spin: Aims Each Part at Player

```
AIDataValue ( "spinHoldTime", Float, ( Val = <#> ))
```

Cluster spins so that each of the different cluster pieces in turn aims its particular weapon at the player, via:

- **`spinHoldTime`** (floating point; default is 5.0) specifies the time (in seconds) holds its orientation after it spins to aim a cluster piece at the player. After the spin hold time is over, Cluster spins again, continuing this process until each of its pieces has had its turn.

Example:

```
AIDataValue ( "spinHoldTime", Float, ( Val = 5.0 ))
```

4.2.16.5 Group Spin: Spins about a Random Axis

Cluster spins around a random axis. Currently, the amount of time it spins is hardcoded at six seconds.

4.2.16.6 Group Linear: Buzz Bomb

```
AIDataValue ( "numBuzzPositions", Integer,  ( Val = <#> ))
AIDataValue ( "buzzHoldTime",     Float,    ( Val = <#> ))
```

Cluster chooses a number of random locations relative to the player and tries to fly to them. It sets its effort level to 3.0 to get to a position, and then reduces it to 1.0 once it has gotten there. Once it gets to a position, it holds its position there for an amount of time. While using Buzz Bomb behavior (both moving to positions and holding at them), Cluster ignores incoming Sinibombs.

Buzz Bomb behavior is controlled by:

- **`numBuzzPositions`** (integer; default is 5) specifies the number of random locations Cluster uses for the Buzz Bomb behavior.
- **`buzzHoldTime`** (floating point; default is 5.0) specifies the time (in seconds) Cluster remains at a buzz bomb position.

Example:

```
AIDataValue ( "numBuzzPositions", Integer,  ( Val = 5 ))
AIDataValue ( "buzzHoldTime",     Float,    ( Val = 5.0 ))
```

4.2.16.7 Group Linear: Ramming

Cluster may ram the player. It uses the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`) for this, with its effort level set to 3.0 while ramming.

4.2.16.8 Group Behavior: Anti-Sinibomb Defenses

If Cluster detects a Sinibomb (via the Anti Missile Detector common boss behavior, `antiMissileSensor`, `onlyDetectSinibombs`) while it is ramming the player, it tries to run away from the Sinibomb. It uses the Fleeing common boss behavior (`fleeDistance`), with its effort level set to 5.0 while fleeing.

4.2.16.9 Group Behavior: Split into Pieces

After being hit by three Sinibombs (this number is currently hardcoded), Cluster splits into its five (or however many are remaining) individual pieces.

4.2.16.10 Individual Behavior: Aim and Snipe

When cluster is divided, each cluster piece circles the player using the Circling the Player or a Location common boss behavior (`circleDistance`). Every five seconds, the klown commands a cluster piece is commanded to stop and aim at the player. Five seconds later, that cluster piece either starts circling again (75% chance) or uses the Lunging common boss behavior (25% chance; `lungeDistance`, `lungeAngle`), while the klown commands another cluster pieces to stop and aim.

4.2.16.11 Individual Behavior: Multi-Ram

For the other attack Cluster can make while divided, each cluster piece circles the player using the Circling the Player or a Location common boss behavior (`circleDistance`). Every five seconds, the klown commands a cluster piece to set its effort level to 3.0 and to ram the player using the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`). After the ram, the cluster piece returns to effort level 1.0 and either starts circling again (75% chance) or uses the Lunging common boss behavior (25% chance; `lungeDistance`, `lungeAngle`).

4.2.16.12 Individual Behavior: Reforming the Group

While Cluster is split up, it will do two attacks in sequence and then reform.

4.2.17 No EXCESSION_16

Level 16 does not have a boss.

4.2.18 EXCESSION_17 “The Lightbulb”

- *Klown Code File:* exc17.fsm
- *Klown:* DEvil_17
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, Scorable, Avoidance, SetTarget, Weapons, SelectiveDamage, SearchForTarget, TargetOffset, TractorAntiMissile, ExcAlive, ExcSound, Attached, ExcPain, TractorBeam2, Boss17MainWeapon
  —*Has Internal Klown:* NormalBehavior

Behaviors for this boss include:

- Initially moves out to 800 meters from the gate.
- Has a selective damage point.
- Has breathing and teeth articulations.
- Can run away from Sinibombs.
- Has a charging photon weapon, which it fires on the player.
- Has a tractor beam, which it can throw asteroids at the player.
- Can also run away from, lunge at, and circle the player.

4.2.18.1 Selective Damage

This boss has a selective damage point and can only be killed there. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`) and currently takes double damage when hit.

4.2.18.2 Breathing

`AIDataValue ( "Breath",  Resource,  ( Val = ( rtSequence, <resource ID> )))`

This boss breathes, and the sequence it plays on its joints while breathing is controlled by:

- **`Breath`** (resource; no default) specifies the `rtSequence` resource that plays as the boss breathes.

Example:

```
AIDataValue ( "Breath",  Resource,  ( Val = ( rtSequence, EXC17_SEQ_MORPH_BREATH )))
```

4.2.18.3 Teeth

`AIDataValue ( "Teeth",  Resource,  ( Val = ( rtSequence, <resource ID> )))`

This boss’s teeth are articulated and are controlled by:

- **`Teeth`** (resource; no default) specifies the `rtSequence` resource that plays as the boss breathes.

Example:

```
AIDataValue ( "Teeth",    Resource,    ( Val = ( rtSequence, EXC17_SEQ_MORPH_TEETH )))
```

4.2.18.4 Anti-Sinibomb Defenses

```
AIDataValue ( "RunAwayDistanceOn",   Float,    ( Val = <#> ))
```

Note: This parameter was originally called `RunAwayDistanceActivate`, but the name was too long and has been shortened to its current form.

The boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`) to detect Sinibombs. When the boss detects an incoming Sinibomb, it runs away if it is far enough from the Sinibomb:

- **`RunAwayDistanceOn`** (floating point; default is 1000.0) is the minimum distance (in meters) to an incoming Sinibomb at which the boss runs away from the Sinibomb. (If the Sinibomb is inside this distance, the boss assumes it cannot get away and ignores the Sinibomb.)

The boss runs away using its Running Away & Returning behavior. Note that when the boss runs away, even from a Sinibomb, it assumes its Throwing Roids behavior when it finishes running away.

Example:

```
AIDataValue ( "RunAwayDistanceOn",   Float,    ( Val = 1000.0 ))
```

4.2.18.5 Tractor Beam

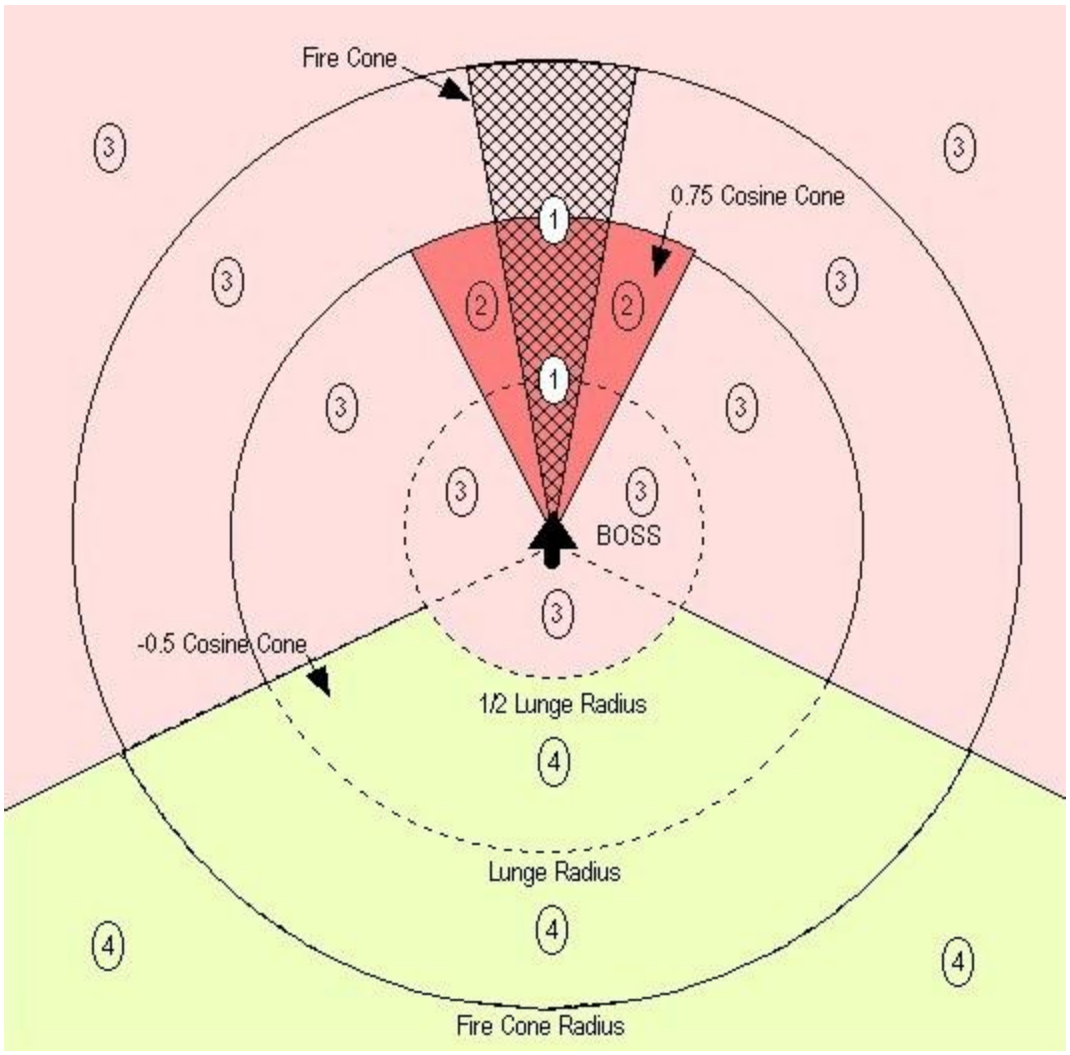
The boss uses the Tractor Beam common ship parameters for its tractor beam (`tractorBeam` and up to 21 more parameters; see Tractor Beam 2).

4.2.18.6 Charging Photon Weapon

- *Klown Code File:* exc17.fsm
- *Klown:* Boss17MainWeapon
  —*Type:* Submachine
  —*Uses External Klowns:* ChargingMainWeapon, PollFast

The boss has a charging photon weapon, which uses the Charging Main Weapon common boss behavior (`mainWeaponCharge`, `mainWeaponChargeTime`). The weapon uses the Fire Cone common ship behavior (`fireConeDistance`, `fireConeAngle`) to govern where it can fire. In addition to all this, the weapon has its own klown controlling its behavior.

4.2.18.7 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Hatched Zone: If the player is within the fire cone radius and fire cone and the boss's main weapon is charged, the boss fires its main weapon (100% chance). Note that the red and pink zones overlap the hatched zone. The hatched zone behavior takes precedence over the red zone and pink zone behaviors. However, if the boss's main weapon is not charged, the hatched zone is ignored and the red zone and pink zone behaviors are used instead.	
2	Red Zone: If the player is within the lunge radius and lunge cone (and, if also inside the hatched zone, the boss's main weapon is not charged), the boss lunges at the player (100% chance).	
3	Pink Zone: If the player is within the pink zone (and, if also inside the hatched zone, the boss's main weapon is not charged), the boss randomly chooses:	
		To move toward the player (50% chance).
		To run away, grab an asteroid, throw it at the player, and return (50% chance).

4	Yellow Zone: If the player is outside one half the lunge radius and is within the -0.5 cosine cone, the boss randomly chooses:	
		To turn toward the player (40% chance).
		To circle the player for a time, then grab an asteroid and throw it at the player (30% chance).
		To circle the player for a time (30% chance).

4.2.18.8 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else.

4.2.18.9 Running Away & Returning

```
AIDataValue ( "RunAwayDistanceGoal",   Float,   ( Val = <#> ))
```

When the boss runs away, it uses:

- **`RunAwayDistanceGoal`** (floating point; default is 1500.0) is the distance (in meters) the boss runs away from the player's position (the position at the instant the behavior is triggered). However, the boss is hardcoded to ensure that it will not run away to a position that is beyond 7,000 meters of the center of the level.

The boss has its effort level set to 2.5 while running away. When done running away, it has its effort level reset to 1.0 and immediately switches its Throwing Roids behavior. After the boss throws a roid, it returns to its home position and resumes its normal attack behaviors. While returning home, the boss has its effort level set to 2.5.

Example:

```
AIDataValue ( "RunAwayDistanceGoal",   Float,   ( Val = 1500.0 ))
```

4.2.18.10 Throwing Roids

```
AIDataValue ( "roidDistance",   Float,   ( Val = <#> ))
AIDataValue ( "roidCosAngle",   Float,   ( Val = <#> ))
```

When the boss wants to throw a roid, it first must find a roid, using its roid sensor and looking in the cone defined by:

- **`roidDistance`** (floating point; default is 1500.0) is the maximum distance (in meters) the boss looks for roids.
- **`roidCosAngle`** (floating point; default is 0.0) is the cosine (see Quick Table of Cosines) of the angle that with the `roidDistance` forms the cone in which the boss searches for roids.

The boss searches for roids in its roid cone and chooses the closest roid it finds in the cone. The boss grabs the roid using its tractor beam and throws the roid at the player. After throwing the roid, boss returns home (see Running Away & Returning).

Notes:

1. If the boss cannot find a roid in its roid cone, it abandons the Throwing Roid behavior, returns home (if necessary), and reassumes its normal attack behavior.

2. If the boss grabs a roid that is somehow destroyed before the boss throws it (such as the boss accidentally damaging and destroying it), the boss abandons the Throwing Roid behavior, returns home (if necessary), and reassumes its normal attack behavior

Example:

```
AIDataValue ( "roidDistance",   Float,    ( Val = 1500.0 ))
AIDataValue ( "roidCosAngle",   Float,    ( Val =    0.0 ))
```

4.2.18.11 Lunging

The boss can lunge at the player using the Lunging ("Standard Attack") common boss behavior (`lungeDistance`, `lungeAngle`).

4.2.18.12 Circling

The boss can circles the player, using the Circling the Player or a Location common boss behavior (`circleDistance`). The boss first circles for 20 ticks (1 tick = 0.05 seconds; 1 second = 20 ticks), then stops and aims at the player for 20 ticks, and then either chooses to throw a roid at the player (50% chance) or choose an attack behavior based on the player's position (50% chance).

4.2.19 EXCESSION_18 “Brain”

- *Klown Code File:* exc18.fsm
- *Klown:* DEvil_18
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, SpinEngines, Attached, SetTarget, FireControl, Weapons, SearchForTarget, ExcAlive, ExcSound, Scorable, Skeleton, SelectiveDamage, Master, TargetOffset, AntiMissileDetector, BlastWave
  —*Has Internal Klown:* NormalBehavior

Behaviors for this boss include:

- Initially moves out to 800 meters from the gate.
- Has three waypoints between which it flies.
- When it reaches a new way point, it lays warrior eggs.
- If the player is near, it does one of three attacks: it circles the player and fires slow missiles, it lays another egg, or it emits blast waves.
- If the player is far, it circles the waypoint and launches slow missiles, or lays more eggs.
- When the boss detects a Sinibomb, it either blasts it away with the blast wave, or activates shields.

4.2.19.1 Selective Damage

This boss has a selective damage point on his bulging sack. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.19.2 Shield Generator Turret

`AIDataValue ( "shieldRechargeTime ", Float,    ( Val = <#> ))`

This boss has a shield generator turret, which allows shields to be thrown up when an incoming Sinibomb is detected. The shield generator turret is attached to the boss, and the shield turret ***must*** be labeled with `Tag = 1`. The turret can be destroyed, at which point, the shields can no longer be activated. The following parameter controls the shield:

- **`shieldRechargeTime`** (floating point; default is 15.0) specifies the amount of time (in seconds) it takes to recharge the shield once the shield is overwhelmed with damage.

Note: Also use the `shieldContainer` parameter (see StartWithShield) to specify the resource ID of the shield the boss uses. In addition, to change the shield thrown up by the turret, you must modify the shield turret resource, `EXC_SHIELD_GENERATOR` in turret.r. The shield generator’s `rtAI` also contains a `shieldContainer` parameter that specifies resource ID for the shield’s container.)

Example:

```
defres rtAI ( EXCESSION_18 )
(
   ...
AIDataValue ( "shieldRechargeTime ", Float,    ( Val = 15.0 ))
AIDataValue ( "shieldContainer",     Integer,  ( Val = EXC_SHIELD ))
   ...
)
```

```
defres rtAI ( EXC_SHIELD_GENERATOR )
(
   ...
AIDataValue ( "shieldContainer",    Integer,  ( Val = EXC_SHIELD ))
   ...
)
```

4.2.19.3 Weaponry

This boss has a blast wave weapon that it uses to attack the player and as an anti-Sinibomb defense. The boss has a blast sensor for the weapon, which uses the Blast Wave common boss behavior (`blastDamage` and the other nine blast-wave parameters). The blast weapon must be charged before it can be used (this has its own charging code and does not use the Charging Main Weapon common boss behavior). *Note:* When the boss gets hit by a Sinibomb, its blast wave recharges immediately.

The boss also has a gun that fires "slow but deadly" missiles.

4.2.19.4 Anti-Sinibomb Defenses

The boss uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `onlyDetectSinibombs`) to detect Sinibombs. When the boss detects an incoming Sinibomb, it can either blasts it away with its blast wave or activates shields:

- If both the blast wave and shield are charged up, the boss randomly chooses one (50% chance for each) to use.
- If only the blast wave is charged up, the boss uses the blast wave.
- If only the shield is charged up, the boss uses the shield.
- If neither the blast wave nor the shield is charged up, the boss uses no Sinibomb defense.

4.2.19.5 Waypoints

```
AIDataValue ( "waypoints",           Resource, ( Val = ( rtVectorArray, <resource ID> )))
AIDataValue ( "waypointSpeed",       Float,    ( Val = <#> ))
AIDataValue ( "maxDistanceFromHome", Float,    ( Val = <#> ))
AIDataValue ( "wayPointTime",        Float,    ( Val = <#> ))
```

The boss moves to waypoints using these parameters:

- **`waypoints`** (resource; default is 1) sets the waypoints to which the boss travels. It points to an `rtVectorArray` resource named *<resource ID>*, which is the list of the waypoints (in world coordinates) for the boss. There can be any number of waypoints in the list. When the boss reaches the last waypoint in the list, it starts over again, traveling to the first waypoint. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtVectorArray` resources.
- **`waypointSpeed`** (floating point; default is 1.0) controls how quickly the boss moves between waypoints. Enter a floating point value for *<#>*; this is a multiplier value that is applied to the standard motion parameters described in the `rtBodyMotion` and `rtBodyControl` resources for the boss.
- **`maxDistanceFromHome`** uses the Maximum Distance from Home common boss behavior and controls the maximum distance the boss moves from a waypoint while pausing there. (The boss is hardcoded to reset its

home position to its current waypoint.) Enter a floating point value for <#>; this is maximum distance in meters that the boss moves from the waypoint.

- **wayPointTime** (floating point; default is 60.0) specifies the amount of time (in seconds) the boss remains in the vicinity of a waypoint before moving on to the next waypoint.

Example:

```
AIDataValue ( "waypoints",           Resource, ( Val = ( rtVectorArray, EXCESSION_18 )))
AIDataValue ( "waypointSpeed",       Float,    ( Val =    5.0 ))
AIDataValue ( "maxDistanceFromHome", Float,    ( Val = 1200.0 ))
AIDataValue ( "wayPointTime",        Float,    ( Val =   60.0 ))

defres rtVectorArray ( EXCESSION_18 )
(
   Vec[] = ( Value = (     0.0, -2000.0, 0.0 ))
   Vec[] = ( Value = (  1500.0,  1200.0, 0.0 ))
   Vec[] = ( Value = ( -1500.0,  1200.0, 0.0 ))
)
```

4.2.19.6 Player Proximity

```
AIDataValue ( "enemyIgnoreDistance",   Float, ( Val = <#> ))
AIDataValue ( "enemyTooCloseDistance", Float, ( Val = <#> ))
```

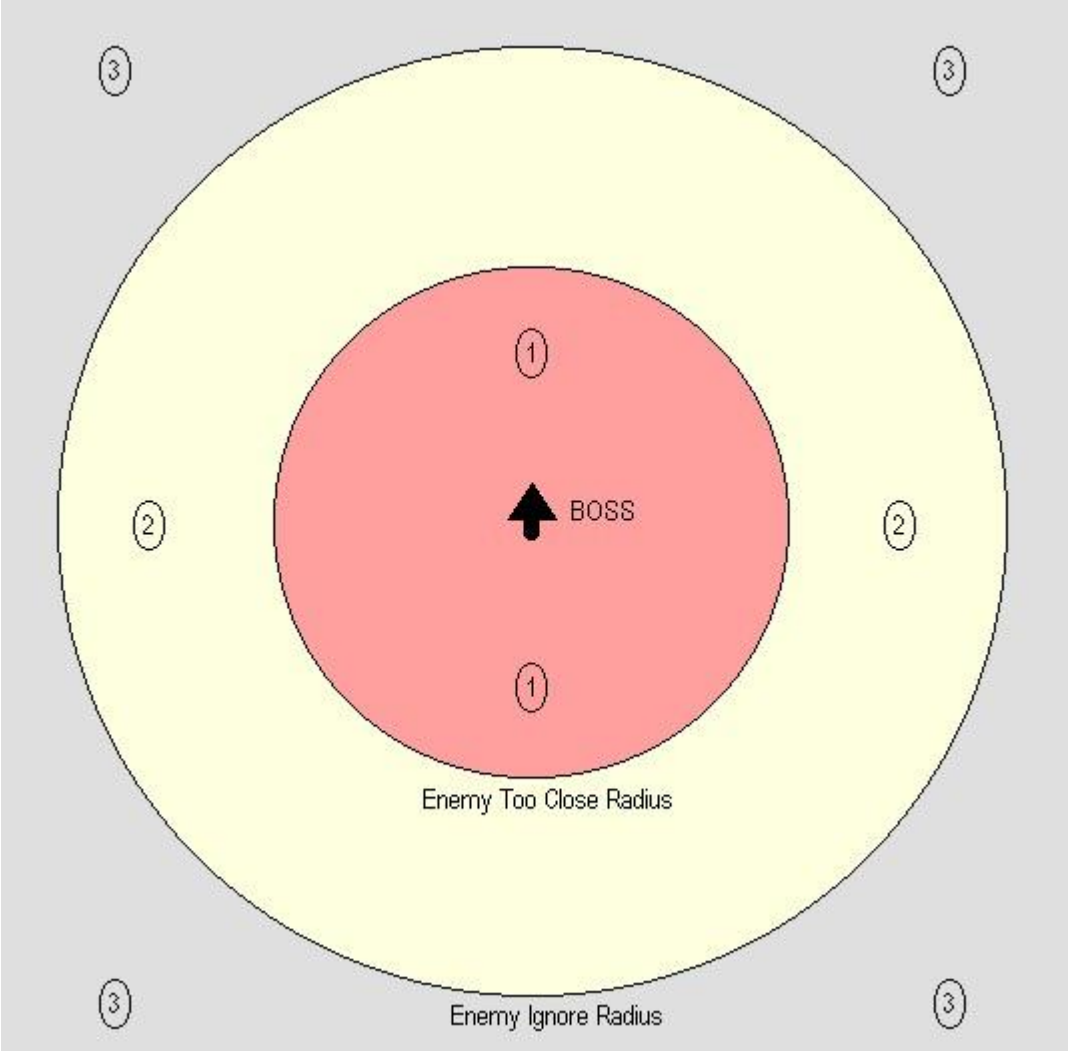
Two parameters that influence the boss's behaviors based on player proximity:

- **enemyIgnoreDistance** (floating point; default is 1500.0) is the distance (in meters) to the player at or beyond which the boss uses "player too far" behaviors instead of "player close" behaviors.
- **enemyTooCloseDistance** (floating point; default is 450.0) is the distance (in meters) to the player at or within which increases the probability that the boss will emit a blast wave.

Example:

```
AIDataValue ( "enemyIgnoreDistance",   Float, ( Val = 1500.0 ))
AIDataValue ( "enemyTooCloseDistance", Float, ( Val =  400.0 ))
```

4.2.19.7 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the enemy too close radius, the boss randomly chooses:	
		To make a circling attack (50% chance if blast not recharged; 16.67% chance if blast recharged).
		Lay an egg (50% chance if blast not recharged; 16.67% chance if blast recharged). *Note:* Even if the boss already has laid all its planned eggs for the current waypoint, it still lays an the egg for this action.
		Fire its blast wave (0% chance if not recharged; 66.67% chance if recharged).
2	Yellow Zone: If the player is within the enemy ignore radius but is outside the enemy too close radius, the boss randomly chooses:	
		To make a circling attack (50% chance if blast not recharged; 33.33% chance if blast recharged).
		Lay an egg (50% chance if blast not recharged; 33.33% chance if blast recharged). *Note:* Even if the boss already has laid all its planned eggs for the current waypoint, it still lays an the egg for this action.
		Fire its blast wave (0% chance if not recharged; 33.33% chance if recharged).

3	Gray Zone: If the player is outside the too far radius, the boss randomly chooses:	
		To make a circling attack (50% chance if all planned eggs are laid at waypoint; 33.33% chance if not all planned eggs are laid yet).
		Circle waypoint (50% chance if all planned eggs are laid at waypoint; 33.33% chance if not all planned eggs are laid yet).
		Lay an egg (0% chance if all planned eggs are laid at waypoint; 33.33% chance if not all planned eggs are laid yet).

4.2.19.8 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else. It then assumes its attack behaviors and moves toward the player.

4.2.19.9 Egg Laying

`AIDataValue ( "numEggsPerAttack", Integer, ( Val = <#> ))`

The boss will lay at most one batch of eggs at each waypoint. It will spin in place as it does so, scattering the eggs in all directions. The number of eggs laid are controlled by:

- **`numEggsPerAttack`** (integer; => 0; default is 3) sets the number of eggs to lay at a waypoint.

Example:

```
AIDataValue ( "numEggsPerAttack", Integer, ( Val = 5 ))
```

The weapon `EXW_EGGLAYER_WARRIOR` (in missile.r) contains the weapon used by the boss to lay the eggs. The weapon must use the `GUNCAT_MINES` gun category. The firing sequence contains the mesh morph for the boss and describes what egg is laid. The egg `EXS_WARRIOR_EGG` has an `rtSequence` resource that specifies how long the egg takes to hatch and what kind of enemy hatches from the egg. (Note that the `rtEmitInfo` action for the hatchling starts only after the model scaling action for the egg is complete. Changing the times for the model scaling action thus changes the incubation time for the egg.)

The egg itself uses an "Entity" klown, which does nothing but accept damage. The health of the egg can be set in the egg's `rtAI` property. See Egg for more details.

4.2.19.10 Circling

The boss occasionally circles its waypoint, using the Circling the Player or a Location common boss behavior (`circleDistance`).

4.2.19.11 Circling Attack

`AIDataValue ( "numMissilesPerAttack", Integer,  ( Val = <#> ))`

This boss will on occasion circle either its waypoint (if the player is outside the `enemyIgnoreDistance`) or the player, and fire slow missiles. It uses the Circling the Player or a Location common boss behavior (`circleDistance`) to accomplish this. Each time this attack is done, it will fire a number of missiles before moving on to a different attack., per the following:

- **numMissilesPerAttack** (integer; default is 3) specifies the number of missiles the boss fires when making a circling attack.

Example:

```
AIDataValue ( "numMissilesPerAttack", Integer,  ( Val = 4  ))
```

These are slow missiles and must be fired by the weapon using GUNCAT_MAIN.

4.2.19.12 Pausing

```
AIDataValue ( "secsBetweenBehaviors",       Float,    ( Val = <#> ))
```

This boss pauses between its attacks. During a pause, it aims at the player and launches slow missiles. The pause time is controlled by the following parameter:

- **secsBetweenBehaviors** (floating point; default is 2.0) specifies the time (in seconds) the boss pauses between attacks.

Example:

```
AIDataValue ( "secsBetweenBehaviors",       Float,    ( Val = 2.0 ))
```

If the boss's shield is destroyed, the boss will no longer pause between attacks. Instead, it starts its next attack right after it completes the previous one.

4.2.20 EXCESSION_19 “Twins”

- *Klown Code File:* exc19.fsm
- *Klown:* DEvil_19
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, BarrelRollEngines, Attached, SetTarget, Weapons, SearchForTarget, ExcAlive, ExcSound, Attached, LaunchMother, LaunchChild, TargetOffset, Scorable, Skeleton, SelectiveDamage, BlastWave, AntiMissileDetector, Boss19MainWeapon, Boss19Guts

- *Klown Code File:* exc19.fsm
- *Klown:* Boss19Guts
  —*Type:* Submachine

The player can face up to two `EXCESSION_19` bosses, the Twins, on one level (one arrives a few seconds after the first). They can fight the player separately or combine to make powerful attacks.

One level has only a single `EXCESSION_19` boss. This boss uses the `EXCESSION_19`’s behaviors, but since there is only one twin present, it always uses the twin-alone attack patterns.

The behaviors for `EXCESSION_19` are:

- One `EXCESSION_19` arrives from each gate.
- One `EXCESSION_19` alone on a level randomly chooses from one of its three attack behaviors to attack the player.
- Two `EXCESSION_19`s that are on a level but are not combined take turns attacking the player, each randomly choosing two its three attack behaviors. After each boss takes its turn, the two bosses try to combine.
- A combined twin `EXCESSION_19` chooses from one of its three attack behaviors to attack the player. When the combined twins fire their charging photon weapons, they achieve a blast effect that isn’t present when fired individually. A combined twin will separate when hit by a Sinibomb.
- If it cannot detect the player (such as if the player is cloaked), the boss parks at the gate.
- If the boss senses an incoming Sinibomb, it flees. However, a combined twin can also use the blast effect of its combined photon weapons against Sinibombs.
- Each twin has a selective damage point on its belly.

4.2.20.1 Selective Damage

Each twin has a selective damage point on its belly and uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.20.2 Charging Photon Weapon

- *Klown Code File:* exc19.fsm
- *Klown:* Boss19MainWeapon
  —*Type:* Submachine
  —*Uses External Klowns:* ChargingMainWeapon, PollFast

Each twin has a charging photon blast as its main weapon. This weapon uses the Charging Main Weapon common boss behavior (`mainWeaponCharge`, `mainWeaponChargeTime`) and emits a blast wave (using the Blast Waves common boss behavior, `blastDamage` and the other nine blast-wave parameters). *Note:* To change the weapon’s

charging time, make sure that the charging time of the photon shot matches that of the Charging Main Weapon common boss behavior.

4.2.20.3 Combined-Twins Blast Wave

- *Klown Code File:* exc19.fsm
- *Klown:* Boss19MainWeapon
  —*Type:* Submachine
  —*Uses External Klowns:* ChargingMainWeapon, PollFast

```
AIDataValue ( "minAttackDistance",   Float,    ( Val = <#> ))
```

The twins, ***only when combined***, has a blast wave weapon (in addition to their individual charging photon weapons) that uses the Blast Wave common boss behavior (`blastDamage` and the other nine blast-wave parameters). The blast wave emanates from the center of the twins and can attack the player and be used as an anti-Sinibomb defense, as described in Common Boss Behaviors. The `minAttackDistance` common ship parameter also controls this weapon:

- **`minAttackDistance`** specifies maximum distance (in meters) from the player that firing the combined-twins blast wave has effect.

Example:

```
AIDataValue ( "minAttackDistance",   Float,    ( Val = 600.0 ))
```

4.2.20.4 Anti-Sinibomb Defenses

Each twin uses the Anti-Missile Detectors common boss behavior (`antiMissileSensor`, `onlyDetectSinibombs`) to detect Sinibombs.

A twin boss's main defense against Sinibombs is to flee (using the Fleeing common boss behavior, `fleeDistance`) at high speed (effort level set to 3.0 while fleeing). Since the twins are fast, however, this is a quite effective tactic, and the boss should be able avoid almost all Sinibombs fired at if from its sides or rear. Typically, only Sinibombs fired head-on at the boss as it is approaching the player have a chance of hitting.

If the twins are combined, they will also use the blast effect of their photon weapons against Sinibombs, if the weapon happens to be recharged when needed for Sinibomb defense.

4.2.20.5 Twins on the Attack

The twins have three attack behaviors: 1) Circle & Fire, 2) Barrel Roll Charge, and 3) Mine-Laying Charge. These are used as follows:

Twin Alone	When ***only one twin is present*** on the level, it randomly chooses one of the three attack behaviors each time it makes an attack.
Twins Apart	When ***both twins are present but are not combined*** on the level, they take turns attack the player. First, one twin will make two attacks on the player (each attack is randomly chosen from the three attacks) while the other twin circles. Then the other twins will make two attack while the first twin circles. After each twin has had its turn, they will attempt to combine (see Mergers and Divestitures).
Twins Combined	When ***the twins are combined with one another*** on the level, they fly in tandem together and make synchronized attacks, randomly chooses one of the three attack behaviors each time they make an attack. The combined twins have a blast wave weapon in addition to the individual

	weapons.

4.2.20.6 Mergers and Divestitures

When both twins are present on the level but are not combined, they try to combine after the pair complete their round of twins apart attacks on the player (see Twins on the Attack). The twins fly out to a common point and try to merge. The point is the closest point to each twin that is their `fleeDistance` away from the player. The player can interrupt the merging process by firing a Sinibomb at either twin, whereupon the twins revert to another round of twins apart attack behavior before trying to merge again.

Once combined, the twins immediately detach from one another (and resume their twins apart attack behavior) when they are hit by a Sinibomb.

4.2.20.7 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 600 meters away before doing anything else.

4.2.20.8 Parking at the Gate

If the boss cannot detect the player (such as if the player is cloaked), the boss parks at the gate using the Park at Gate common boss behavior.

4.2.20.9 Circle & Fire

A twin boss (whether as a separate twin or as both twins combined) using this behavior circles the player and fires its charging photon weapon at the player. Circling uses the Circling the Player or a Location common boss behavior (the `circleDistance` AI data value sets the circling radius), and the boss circles until its main weapon is charged. While the weapon is recharging, the boss circles with its effort level set to 3.0. Once the main weapon is charged, it will stop, aim at the player, and fire the weapon.

4.2.20.10 Barrel Roll Charge

A twin boss (whether as a separate twin or as both twins combined) using this behavior charges the player (before charging, it may first try to get some distance from the player using the Fleeing common boss behavior, `fleeDistance`). The charge uses the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`). As the boss charges toward the player, it recharges its main weapon and fires it. Once within its lunge distance, it no longer tries to avoid Sinibombs. It ends its attack with a barrel roll away from the play, using the BarrelRoll common ship behavior (`barrelSpinEffort`, `barrelStrafeEffort`).

4.2.20.11 Mine-Laying Charge

`AIDataValue ( "minesPerPass", Integer, ( Val = <#> ))`

A twin boss (whether as a separate twin or as both twins combined) using this behavior follows the same basic pattern as the barrel roll charge (see Barrel Roll Charge). However, instead of firing its weapon, once it gets within the Clumsy Charge's lunge distance, it drops a number of sucker mines, which try to hunt down the player. The sucker mines are attached to the boss using the Attached Missiles & Mines common ship behavior. The number of mines dropped off is controlled by the following parameter:

- **`minesPerPass`** (integer; default is 4) specifies he number of mines the boss drops each time it makes a mine laying charge. *Note:* Currently, the boss is hardwired to drop mines in multiples of two, so set `minesPerPass` as a multiple of two.

Example:

```
AIDataValue ( "minesPerPass", Integer, ( Val = 4 ))
```

4.2.21 No EXCESSION_20

Level 20 does not have a boss.

4.2.22 EXCESSION_23 (“Phoenix”)

- *Klown Code File:* exc21.fsm
- *Klown:* DEvil_21
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, RollTowardEngines, SetTarget, FireControl, Weapons, SearchForTarget, MineControl, ExcAlive, ExcSound, Attached, Skeleton, SelectiveDamage, TargetOffset, Scorable, ChargingMainWeapon
  —*Has Internal Klown:* NormalBehavior

(*Note:* The boss is labeled EXCESSION_23 in the resource files, but uses the DEvil_21 klown.)

Behaviors for this boss include:

- The boss has a selective damage point inside its rib cage.
- The boss works like a battleship, with laser-armed spikes along its sides. It circles the player, rolling and firing its broadside.
- If the player is close to the boss, it charges the player, first firing its main lightning gun and then laying mines.
- If it cannot detect the player (such as if the player is cloaked), the boss parks at the gate.

4.2.22.1 Selective Damage

The boss has a selective damage point inside its rib cage. It can only killed on this weak point. This point uses the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.22.2 Charging Lightning Gun

The boss’s main weapon is a charging lightning gun, which uses the Charging Main Weapon common boss behavior (`mainWeaponCharge`, `mainWeaponChargeTime`). The lightning gun must be hooked into the weapon with gun category `GUNCAT_MAIN`. If the boss is hit anywhere (not just at its selective damage point) by a Sinibomb while the weapon is charging, the weapon loses all its charge and must start recharging again.

4.2.22.3 Cutting Laser Spikes

```
AIDataValue ( "laserSweepAngle",        Float,    ( Val = <#> ))
AIDataValue ( "laserSweepTime",         Float,    ( Val = <#> ))
```

The boss has a series of cutting lasers spikes along its sides; it is thus capable of firing a cutting-laser broadside (and does so as one of its attack behaviors). The boss sinusoidally waves its spikes, per the following parameters:

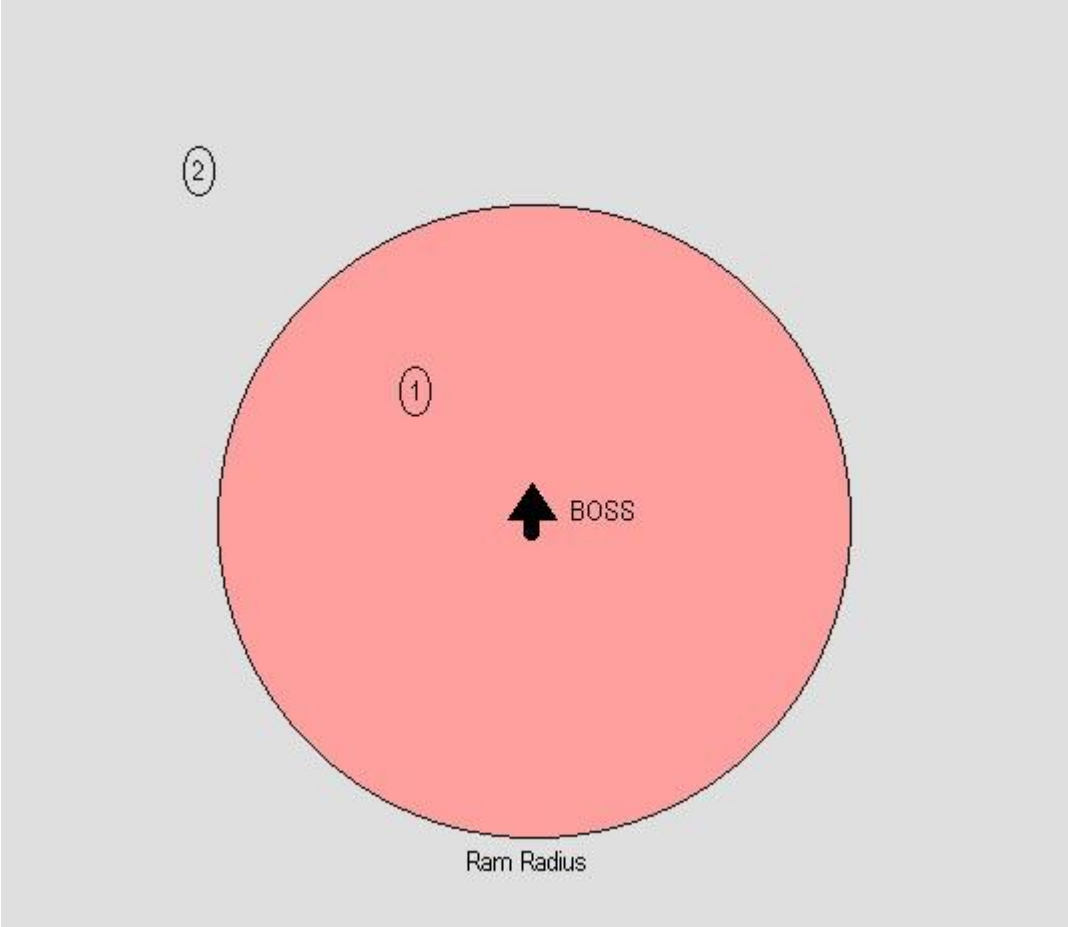
- **`laserSweepAngle`** (floating point; default is 30.0) sets the angle (in degrees) that the cutting lasers are swung up and down.
- **`laserSweepTime`** (floating point; default is 1.0) specifies the time (in seconds) for the round-trip time of the sinusoidal oscillation.

Note: For the spikes to be waved sinusoidally, you must also use the `disableSkeletalPhysics` parameter (see Disabling Skeletal Physics). Otherwise, the skeletal spring system prevents the sinusoidal movement.

Example:

```
AIDataValue ( "laserSweepAngle",        Float,    ( Val = 25.0 ))
AIDataValue ( "laserSweepTime",         Float,    ( Val =  1.0 ))
AIDataValue ( "disableSkeletalPhysics", Integer,  ( Val =  1 ))
```

4.2.22.4 Behaviors Triggered by Player Position



The position of the player triggers various behaviors of the boss:

1	Red Zone: If the player is within the boss's ram radius, the boss chooses:	
		To charge, fire its main weapon, and lay mines (100% chance if main weapon is recharged; 0% chance if main weapon is not recharged).
		To circle and fire its broadsides (0% chance if main weapon is recharged; 100% chance if main weapon is not recharged).
2	Gray Zone: If the player is outside the ram radius, the boss chooses to circle and fire its broadsides (100% chance).	

4.2.22.5 Clears the Gate

Upon arrival in the level, the boss clears the gate, moving 800 meters away before doing anything else. It then assumes its attack behaviors and moves toward the player.

4.2.22.6 Parking at the Gate

If the boss cannot detect the player (such as if the player is cloaked), the boss parks at the gate using Park at Gate.

4.2.22.7 Circle & Fire Broadside

```
AIDataValue ( "circleEnemyTime",     Float,    ( Val = <#> ))
```

The boss will circle the player, firing cutting-laser broadsides at the player, for a period of time. At the end of the period, the boss will select a new behavior, either charging the player if the player is close enough (see Charge, Fire Main Weapon, & Lay Mines) or continuing to circle and firing broadsides. Circling is controlled by the Circling the Player or a Location common boss behavior (`circleDistance`), and rolling is controlled by the Roll Effort common ship behavior (`rollEffort`). The boss has its effort level set to 2.0 when using this behavior. The following parameter further controls this attack behavior:

- **`circleEnemyTime`** (floating point; default is 6.0) specifies the amount of time (in seconds) the boss spends making a circle and fire broadside attack. At the end of that time, it chooses its next attack behavior.

Example:

```
AIDataValue ( "circleEnemyTime",     Float,    ( Val =    6.0 ))
AIDataValue ( "circleDistance",      Float,    ( Val = 1200.0 ))
AIDataValue ( "rollEffort",          Float,    ( Val =    0.2 ))
```

4.2.22.8 Charge, Fire Main Weapon, & Lay Mines

```
AIDataValue ( "ramDistance",      Float,   ( Val = <#> ))
AIDataValue ( "fireDelay",        Float,   ( Val = <#> ))
```

When the boss selects an attack behavior, it will charge the player if the player is close to the boss. It uses the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`) for its charge, and it fires its main weapon, the lightning gun, on the player (see Circle & Fire Broadside). At the end of its charge, it lays mines, using the Mine Laying common boss behavior (`maxMines`, `minMines`, `mineCategory`). Beside the common boss behavior parameters, the following parameters control this behavior:

- **`ramDistance`** (floating point; default is 1.0) specifies the maximum distance (in meters) from the player at which the boss decides to make this charging attack behavior.
- **`fireDelay`** (floating point; default is 1.0) specifies the amount of time (in seconds) the boss delays in firing its main weapon (the lightning gun) after starting its charging attack behavior.

Note: When the boss rams the player, it spits the player out of its maw to prevent bad collisions.

Example:

```
AIDataValue ( "ramDistance",      Float,   ( Val = 1100.0 ))
AIDataValue ( "fireDelay",        Float,   ( Val =    1.0 ))
```

4.2.23 EXCESSION_24 (“The Sinistar”)

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Team, Collision, MoveEngines, OrientEngines, Avoidance, PollFast, CircleEnginesNoSpin, SetTarget, FireControl, Weapons, SearchForTarget, ExcAlive, ExcSound, Attached, Skeleton, SelectiveDamage, TargetOffset, Scorable, DefenseGrid, DEvil_24_Flaps, DEvil_24_Eyes, DEvil_24_Mouth, DEvil_24_SuckerMines, DEvil_24_EggLayer, DEvil_24_BlowOut, Birth
  —*Has Internal Klown:* NormalBehavior

Behaviors for this *bosso di tutti bossi* include:

- Selective damage; the boss has only two points on it that are vulnerable to Sinibombs.
- Anti-Sinibomb flaps, which most of the time cover its vulnerable points.
- Inertialess movement, allowing it to move like a UFO.
- Eye weapons, five different weapons that can fire out its eyes.
- Long-range missiles it uses to make the player miserable.
- Six orbitals that orbit the boss and shoot at the player.
- Egg laying ability, with each egg hatching into a boss.
- When hurt sufficiently, blasts everything out from the level except the player for a final showdown.

4.2.23.1 Inertialess Movement

This boss travels like an inertialess UFO. It can move equally well in any direction without turning, decelerating, accelerating, etc. While moving, it rotates to keep facing the player. This rotation rate should slightly lag behind the player’s ability to flank the boss.

4.2.23.2 Selective Damage

This boss has two weak points, one under each of its flaps, that are its only points vulnerable to Sinibombs. These use the Selective Damage common boss behavior (`jointSelectiveDamage`).

4.2.23.3 Anti-Sinibomb Flaps

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_Flaps
  —*Type:* Submachine

```
AIDataValue ( "flapsOpenSeq",      Resource, ( Val = ( rtSequence, <#> )))
AIDataValue ( "flapsCloseSeq",     Resource, ( Val = ( rtSequence, <#> )))
AIDataValue ( "flapsBreatheTime",  Float,    ( Val = <#> ))
AIDataValue ( "flapsOpenTime",     Float,    ( Val = <#> ))
```

The Sinistar has only one defense against Sinibombs, but it is a foolproof one. Armored flaps invulnerable to Sinibombs cover its selective damage points, and prevent Sinibombs from damaging the boss at all. These flaps only open under certain conditions, which are the only times the Sinistar can take damage:

- **Breathing:** The boss must "breathe" at periodical intervals, briefly opening its flaps to do so.
- **Sinibomb Reaction:** If the flaps are closed when the boss is hit by a Sinibomb, it will briefly open its flaps in reaction. (Thus, at great crystal expense, the player can stagger Sinibombs so that one causes the flaps to open and the following one strikes a vital point.)
- **Taunting:** The boss opens its flaps briefly when it does a full-stop taunt of the player. (Taunting is covered below in its own section.)

The parameters that control the flaps and breathing are:

- **`flapsOpenSeq`** (resource; no default) specifies the `rtSequence` resource that opens the flaps.
- **`flapsCloseSeq`** (resource; no default) specifies the `rtSequence` resource that closes the flaps.
- **`flapsBreatheTime`** (floating point; default is 60.0) specifies the time (in seconds) the boss goes between taking breathes. Its flaps are closed during this time.
- **`flapsOpenTime`** (floating point; default is 60.0) specifies the time (in seconds) the boss must have its flaps open while it breathes.

Example:

```
AIDataValue ( "flapsOpenSeq",        Resource, ( Val = ( rtSequence, EXC24_SEQ_OPENFLAPS
                ) ))
AIDataValue ( "flapsCloseSeq",       Resource, ( Val = ( rtSequence, EXC24_SEQ_CLOSEFLAPS
                ) ))
AIDataValue ( "flapsBreatheTime",    Float,    ( Val = 60.0 ))
AIDataValue ( "flapsOpenTime",       Float,    ( Val = 15.0 ))
```

4.2.23.4 Sinibomb Damage Reaction

Whenever the boss takes damage from a Sinibomb (i.e., a Sinibomb that hits a selective damage point when the flaps are open), the boss charges the player. This uses the Charge behavior described below.

4.2.23.5 Eye Weapons

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_Eyes
  —*Type:* Submachine
  —*Uses External Klowns:* Attached

```
AIDataValue ( "eyesChargeTime",     Float,    ( Val = <#> ))
AIDataValue ( "eyesDischargeTime",  Float,    ( Val = <#> ))
AIDataValue ( "eyeEffectContainer", Integer,  ( Val = <resource ID> ))
AIDataValue ( "eyeWeaponColors",    Resource, ( Val = ( rtIntegerArray, <resource ID> )))
AIDataValue ( "turretList",         Resource, ( Val = ( rtTurretInfo, <resource ID> )))
```

These five weapons emanate from the two "eyes" of the boss. When the boss fires an eye weapon, it randomly (with equal probability) selects one of the allowed weapons (some behaviors, like Seek & Strafe, exclude some eye weapons) and fires it from both eyes (i.e., emits two shots, one from each eye). The boss has five distinctly different eye weapons:

- **Tractor Beams:** This pulls and holds for a time the player ship within range of the boss, which releases seeking mines to attack the ship. See Tractor Attack for details.
- **Gattling Laser:** This is a laser with an extremely high rate of fire but with sub-par accuracy.
- **Cutting Beam.**

- **Concussion Burst.**
- **Lightning Gun.**

The five eye weapons use the Attached Turret common ship behavior. Each weapon must have a tag, and each different weapon type must have a different tag from the other weapon types.

The eyes have a charging effect for each of these weapons. An effect container (specified by `eyeEffectContainer`) is emitted at each place on the model labeled by the model attribute named "`effect`". The eye effect container is a monochromic white lens flare. Depending on the weapon, the effect is color modulated based on the colors passed in via `eyeWeaponColors`. The eye weapon colors lists the colors for each of the five eye weapons. The first item in the list is the color to use for the turrets with tag 0, the second item in the list is the color to use for turrets with tag 1, etc. The eyes have a finite charge and discharge time. During these times, the eye effect containers fade in and out. If the eye effect containers have attached lights, the light intensity can be raised and lowered.

The following parameters control the eye weapons:

- **`eyesChargeTime`** (floating point; default is 3.0) specifies the time (in seconds) it takes to charge an eye weapon.
- **`eyesDischargeTime`** (floating point; default is 2.0) specifies the time (in seconds) it takes to discharge an eye weapon.
- **`eyeEffectContainer`** (resource ID; default is 1) specifies the resource ID of the effect container for the boss's eye weapons.
- **`eyeWeaponColors`** (resource; no default) specifies an `rtIntegerArray` resource for the weapons' colors. The `rtIntegerArray` resource has one integer array for each weapon type, with the array specifying the weapon's RGB color value. This color is used to modulate the eye's lens flare as the weapon charges and discharges.
- **`turretList`** (resource; no default) specifies an `rtTurretInfo` resource that lists the eye's weapons. *Note:* Make sure you list the weapons in this list in the same order as you listed the weapon's colors in the `rtIntegerArray` resource for `eyeWeaponColors`.

See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how to define `rtIntegerArray` and `rtTurretInfo` resources.

Note: An earlier AI data value, `eyeSwitchDistance`, is no longer used.

Example:

```
AIDataValue ( "eyesChargeTime",     Float,    ( Val = 2.0 ))
AIDataValue ( "eyesDischargeTime",  Float,    ( Val = 0.5 ))
AIDataValue ( "eyeEffectContainer", Integer,  ( Val = EXC24_EYE_EFFECT ))
AIDataValue ( "eyeWeaponColors",    Resource, ( Val = ( rtIntegerArray, EXC24_EYE_EFFECT
                ) ))
AIDataValue ( "turretList",         Resource, ( Val = ( rtTurretInfo, EXCESSION_24 )))

defres rtIntegerArray ( EXC24_EYE_EFFECT )
(
   Int[] = ( Value = RGBCOLOR( 255,   0,   0 ) )
   Int[] = ( Value = RGBCOLOR( 255, 255,   0 ) )
   Int[] = ( Value = RGBCOLOR(   0, 255,   0 ) )
   Int[] = ( Value = RGBCOLOR(   0,   0, 255 ) )
   Int[] = ( Value = RGBCOLOR( 255,   0, 255 ) )
)
```

4.2.23.6 Long-Range Missiles

The boss will have four missile ports from which it can launch long-range missiles. As of the time this document was written, these were not operational yet.

4.2.23.7 Orbitals

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_DefenseGrid
  —*Type:* Submachine

```
AIDataValue ( "defenseGridTurretTag", Integer,  ( Val = <#> ))
AIDataValue ( "defenseGridMaxAttack", Integer,  ( Val = <#> ))
AIDataValue ( "defenseGridPhiOffset", Float,    ( Val = <#> ))
```

The boss has orbitals buzzing around it as its defense grid. The orbitals all must have the same tag, which is specified using `defenseGridTurretTag`. The maximum number of orbitals that can simultaneously attack is specified by `defenseGridMaxAttack`. The orbitals always try to form up in a circular formation. The circle lies on a sphere centered in the middle of the boss. The size of the circle is specified as an angle (`defenseGridPhiOffset`) from the center of the circle to the actual points on the circle. (Another way to view this is to think about the circle generated by rotating the Z axis about any vector in the X-Y plane by the angle `defenseGridPhiOffset`.) The parameters are:

- **`defenseGridTurretTag`** (integer; default is 1) specifies the tag the orbitals must have in common.
- **`defenseGridMaxAttack`** (integer; default is 3) specifies the maximum number of the orbitals that can attack the player simultaneously.
- **`defenseGridPhiOffset`** (floating point; default is 30.0) specifies the circle the orbitals use to orbit the boss.

Example:

```
AIDataValue ( "defenseGridTurretTag", Integer,  ( Val = 5 ))
AIDataValue ( "defenseGridMaxAttack", Integer,  ( Val = 19 ))
AIDataValue ( "defenseGridPhiOffset", Float,    ( Val = 8.0 ))
```

4.2.23.8 Choosing Player-Oriented Behaviors

```
AIDataValue ( "randomEventTime",     Float,    ( Val = 20 ))
```

After a set amount of time, the boss will randomly choose one of its six player-oriented behaviors: Seek & Strafe, Flee, Charge, Circle, Launch Fighter, or Tractor Attack, unless some other event triggers a behavior. (Egg laying is separate behavior that operates on its own.) The other triggers are: 1) player distance (which triggers taunting), 2) damage from a Sinibomb (which triggers a charge on the player), and 3) the hatching of an egg (which triggers the boss fleeing and letting its hatchling fight the player).

When the boss randomly chooses a normal behavior, the probability of a specific behavior being chosen varies, based on how likely it is to flee (see Fleeing for details). Except for fleeing, all other normal behaviors have equal probability of occurring (except that a tractor attack won't be chosen if the tractor beam is not recharged).

The time interval is controlled by the following:

- **`randomEventTime`** (floating point; default is 1.0) specifies the time (in seconds) that must elapse after the boss chooses a behavior before it can choose another behavior.

Example:

```
AIDataValue ( "randomEventTime",     Float,     ( Val = 20.0 ))
```

4.2.23.9 Behavior Summary

As you may have noticed by now, `EXCESSION_24` has a lot of behaviors. Here's a summary of them:

Flaps	Sinibomb-proof flaps cover the boss's two vulnerable points. The flaps:	
		Open briefly when the boss breathes.
		Open briefly after the boss is hit anywhere by a Sinibomb.
		Open while the boss taunts the player.
		Are closed at all other times.
Orbitals	Are always active in a defense grid circling the boss; can attack the player regardless of the boss's current behavior.	
Egg Laying	The boss lays one egg every so often:	
		If the egg gets destroyed before hatching, the boss lays another egg when it's time to.
		If the egg hatches, the boss ceases egg laying, flees the player, and lets the hatchling go fight.
		If the player approaches the boss (gets within the boss's `maxAttackDistance`) while the boss is fleeing, the boss immediately resumes its player-oriented behaviors (see below), with the chance of fleeing again reduced.
		If the hatchling dies, the boss immediately resumes its player-oriented behaviors and resumes egg laying (with the egg timer reset).
Sinibomb Damage Reaction	If the boss takes damage from a Sinibomb (and not just gets hit by a Sinibomb when the flaps are closed), the boss immediately switches to its Charge behavior.	
Taunting	If the player is too far away (beyond the `tauntDistance`), the boss taunts the player. (The boss does not taunt the player while it is fleeing the player, including the time while a hatchling is alive.)	
Level Blast Out	When damage to the boss reaches a certain point, the boss emits a blast wave that clears everything out of the level except the boss and the player.	
Player-Oriented Behaviors	When the boss is not taunting the player or fleeing the player while a hatchling is alive, the boss randomly selects one of the following:	
		Seek and strafe the player.
		Circle and fire on the player.
		Launch a fighter.
		Charge the player.
		Make a tractor attack if the player is within `maxTractorDistance` (catch the player in a tractor beam and launch sucker mines).
		Flee. However, if the player approaches the boss (gets within the boss's `maxAttackDistance`) while the boss is fleeing, the boss immediately resumes its player-oriented behaviors, with the chance of fleeing again reduced.

4.2.23.10 Egg Laying

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_EggLayer
  —*Type:* Submachine

4.2.23.10.a Egg Spacing

```
AIDataValue ( "eggSecondsPerEgg",    Float,    ( Val = <#> ))
```

The Sinistar periodically lays eggs that hatch into bosses. If an egg successfully hatches (the player can destroy eggs with Sinibombs), the boss flees using the Fleeing common boss behavior (`fleeDistance`), stops laying eggs, and lets baby boss do its evil bidding. (If the player approaches within the boss's `maxAttackDistance` while the hatchling is alive, the boss resumes its player-oriented behaviors and most likely will attack the player.) If the hatchling boss dies, the Sinistar resumes egg laying.

The following parameter controls egg laying:

- **eggSecondsPerEgg** (floating point; default is 1.0) specifies the number of seconds that must pass after the Sinistar lays one egg before it can lay another. `eggSecondsPerEgg` is also the time the Sinistar waits after a hatchling boss dies before resuming egg laying.

Example:

```
AIDataValue ( "eggSecondsPerEgg",    Float,    ( Val = 90 ))
```

4.2.23.10.b Egg Gun

```
defres rtAI ( XW24_BOSS_EGG_LAYER )
(
   Type.Klown = ( KlownName = "EggWeapon" )
AIDataValue
   (
      "Category",          Integer,  ( Val = GUNCAT_MINES )
   )
AIDataValue
   (
      "GunportNames",      Resource, ( Val = ( rtGunportNames, XW24_BOSS_EGG_LAYER ))
   )
AIDataValue
   (
      "eggLayingSequence", Resource, ( Val = ( rtSequence, XW24_BOSS_EGG_LAYER ))
   )
AIDataValue
   (
      "eggContainers",     Resource, ( Val = ( rtIntegerArray, XW24_BOSS_EGG_LAYER ))
   )
)
```

To lay the egg, the Sinistar fires the gun, `XW24_BOSS_EGG_LAYER` (in weapon.r). This gun uses a new weapon klown, the "`EggWeapon`." The gun requires the following AI parameters:

- **Category** must be `GUNCAT_MINES`.
- **GunportNames** must have an `rtGunportNames` resource for the gun (currently, this is `XW24_BOSS_EGG_LAYER`).

- **eggLayingSequence** specifies an `rtSequence` resource (currently, `XW24_BOSS_EGG_LAYER`) that is played when the weapon is fired. The `rtSequence` resource itself must specify an `rtEmitInfo` action named "`eggEmitter`", which creates the egg. (The object specified as being emitted in this `rtEmitInfo` resource in the .r file is a dummy container ID and is replaced with the actual boss container ID each time an egg is laid; see below.)
- **eggContainers** specifies an `rtIntegerArray` resource that has a list of the container resource IDs for the bosses that actually hatch from the egg. (These container IDs are inserted into the emitter, replacing the dummy container ID in the .r file). The first time the egg weapon fires, the egg will hatch the first container in the `eggContainers` list. The second time it fires, the egg will hatch the second container in the list, and so on. *Note:* All bosses can be used in list except for the Sinistar boss itself.

See the separate documents: The Sequencing System for details on sequences, Emitters for `rtEmitInfo` resources, and CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm for `rtIntegerArray` resources. See Egg for more details.

Example:

```
defres rtAI ( XW24_BOSS_EGG_LAYER )
(
   Type.Klown = ( KlownName = "EggWeapon" )
AIDataValue
   (
      "Category",          Integer,  ( Val = GUNCAT_MINES )
   )
AIDataValue
   (
      "GunportNames",      Resource, ( Val = ( rtGunportNames, XW24_BOSS_EGG_LAYER ))
   )
AIDataValue
   (
      "eggLayingSequence", Resource, ( Val = ( rtSequence, XW24_BOSS_EGG_LAYER ))
   )
AIDataValue
   (
      "eggContainers",     Resource, ( Val = ( rtIntegerArray, XW24_BOSS_EGG_LAYER ))
   )
)

defres rtIntegerArray ( XW24_BOSS_EGG_LAYER )
(
   Int[] = ( Value = EXCESSION_01 )
   Int[] = ( Value = EXCESSION_02 )
   Int[] = ( Value = EXCESSION_03 )

   Int[] = ( Value = EXCESSION_05 )
   Int[] = ( Value = EXCESSION_06 )
   Int[] = ( Value = EXCESSION_07 )

   Int[] = ( Value = EXCESSION_09 )
   Int[] = ( Value = EXCESSION_10 )
   Int[] = ( Value = EXCESSION_11 )

   Int[] = ( Value = EXCESSION_13 )
   Int[] = ( Value = EXCESSION_14 )
   Int[] = ( Value = EXCESSION_15 )
```

```
   Int[] = ( Value = EXCESSION_17 )
   Int[] = ( Value = EXCESSION_18 )
   Int[] = ( Value = EXCESSION_19 )
)

defres rtSequence ( XW24_BOSS_EGG_LAYER )
(
   Actions[] =
   (
      ActionId = ( rtEmitInfo, XW24_BOSS_EGG_LAYER )
      Name     = "eggEmitter"
      Time     = 0
   )
   Actions[] = ( ActionId = ( rtSoundAction, XW24_BOSS_EGG_LAYER ) Time = 0 )
)
```

4.2.23.11 Taunting

```
AIDataValue ( "tauntDistance",     Float, ( Val = <#> ))
AIDataValue ( "tauntTimePerTaunt", Float, ( Val = <#> ))
```

The boss taunts the player when the player is beyond a specified distance from the boss. (It does not taunt, however, while it is fleeing the player, regardless of the distance.) When the boss taunts the player, it opens its flaps. (Currently, the length of time the has its flaps opens during a taunt is the same as the `flapsOpenTime` for breathing.) The boss will continue to taunt the player until it has done each of the taunts in its `fleeTauntSeq` list, or until the player has come within the boss's `maxAttackDistance`. Then the boss will randomly pick a behavior.

Taunting is controlled by the following parameters:

- **`tauntDistance`** (floating point; default is 1.0) specifies the minimum distance (in meters) between the boss and the player that triggers taunting.
- **`tauntTimePerTaunt`** (floating point; default is 1.0) specifies the time (in seconds) the boss pauses between each taunt on its `fleeTauntSeq` list.

Example:

```
AIDataValue ( "tauntDistance",     Float, ( Val = 2000.0 ))
AIDataValue ( "tauntTimePerTaunt", Float, ( Val =    8.0 ))
```

4.2.23.12 Fleeing

```
AIDataValue ( "fleeMaxWeight",       Float,    ( Val = <#> ))
AIDataValue ( "fleeMaxCrystals",     Float,    ( Val = <#> ))
```

The boss occasionally flees the player (which allows the player to collect crystals), using the Fleeing common boss behavior (`fleeDistance`). The decision to flee is made when `randomEventTime` triggers a behavior change, and the decision is based on the number of crystals the player has, as follows:

- **`fleeMaxWeight`** (floating point; >= 0.0, >= 1.0; default is 1.0) specifies the maximum probability (0.0 = 0% chance, 1.0 = 100% chance) that the boss will flee. The maximum flee probability is used when the player has no crystals, and the flee probability is linearly reduced based on the number of crystals the player has, reaching 0 chance of fleeing at 100 crystals (assuming `fleeMaxCrystals` is not set; see below). For

example, at `Val = 1.0`, the boss will flee 100% of the time when the player has 0 crystals, 50% of the time at 50 crystals (remember that the range is 0 to 100, not 1 to 100), and 0% of the time at 100 crystals. Setting `Val = 50.0` would reduce the probabilities in the above example by half.

- **`fleeMaxCrystals`** (floating point; default is 1.0) specifies the maximum number of crystals at which the boss has 0% chance of fleeing. For example, at `fleeMaxWeight` with `Val = 0.75` and `fleeMaxCrystals` with `Val = 30.0`, the boss will flee 75% of the time when the player has 0 crystals, 50% of the time at 15 crystals, and 0% of the time at 30+ crystals.

Example:

```
AIDataValue ( "fleeMaxWeight",       Float,    ( Val =  0.75 ))
AIDataValue ( "fleeMaxCrystals",     Float,    ( Val = 30.0 ))
```

Should the player try to approach the Sinistar while it is fleeing, it immediately charges the player and permanently reduces its likelihood of ever fleeing again by 75%.

4.2.23.13 Seek & Strafe

The boss seeks the player, chasing down the player, and then strafes the player, firing on the player as it slides past.

While seeking the player, the boss tries to move to within its standard `minAttackDistance` distance of the player. Once it gets close, it picks a tangential direction and strafes around the player, still aiming at the player. It strafes at effort level 2.0, until it gets to its standard `maxAttackDistance` away from the player. Once at `maxAttackDistance` , it powers down it current eye weapon and waits there while its newly-selected eye weapon powers up. While in this state, the boss does not try to select the tractor-beam eye weapon.

4.2.23.14 Circling

```
AIDataValue ( "circleTimePerWeapon", Float,    ( Val = <#> ))
```

The boss circles the player using the Circling the Player or a Location common boss behavior (`circleDistance`). As it circles, it randomly selects and fires for a time one of its eye weapons (other than the tractor beam), via:

- **`circleTimePerWeapon`** (floating point; default is 1.0) specifies the amount of time (in seconds) the boss uses a weapon before switching to another.

Example:

```
AIDataValue ( "circleTimePerWeapon", Float,    ( Val = 8.0 ))
```

4.2.23.15 Launching Fighters

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_Mouth
  —*Type:* Submachine

When using this behavior, the boss launches one Distiller Evil Torturer (`EVIL_WARRIOR4`) out of its mouth. (Both the number and types of ships launched can be set in the boss's resource for this weapon.) The fighter launch is done as a weapon. It fires the weapon with category `GUNCAT_MAIN`, which is expected to emit a fighter from the boss's mouth.

4.2.23.16 Charging

The boss has its effort level set to 2.0 while charging the player. The boss fires its available weapons as it charges and tries to ram the player (this is the only time the boss will try to ram, rather than just strafing past the player). The boss uses the Clumsy Charge common boss behavior (`lungeDistance`, `peelOffAngle`, `chargeOvershoot`) for its charge.

4.2.23.17 Tractor Attack

- *Klown Code File:* exc24.fsm
- *Klown:* DEvil_24_SuckerMines
  —*Type:* Submachine
  —*Uses External Klowns:* PollUpdate

```
AIDataValue ( "maxTractorDistance",  Float,    ( Val = <#> ))
AIDataValue ( "minesPerSec",         Integer,  ( Val = <#> ))
AIDataValue ( "mineLayRange",        Float,    ( Val = <#> ))
AIDataValue ( "tractorTime",         Float,    ( Val = <#> ))
```

When choosing a player-oriented behavior, the boss can choose to make a tractor attack only if the player is within tractor beam range:

- **`maxTractorDistance`** (floating point; default is 1500.0) specifies the maximum distance (in meters) the player can be from the boss for the boss to make a tractor attack.

When making a tractor attack, the boss comes to a full stop and turns on its eye tractors (which are attached turrets). It tractors the player to a specified distance from the boss. (This tractor beam has its own code and does not use the common Tractor Beam 2 parameters.)

Once the player gets within a specified distance of the boss, it starts releasing two mines (currently this number is hardcoded) per a tunable interval of time. Mine laying occur per the standard missile launching resources, while the mines themselves are attached using the Attached Missiles & Mines common ship parameter. After a certain amount of time has passed, the tractor beams are released and the boss chooses a new behavior. The parameters involved with this behavior are:

- **`mineLayRange`** (floating point; default is 1.0) specifies the distance (in meters) from the boss to which the tractor beams drag the player. (This is also the distance at which the boss starts laying mines.)
- **`minesPerSec`** (integer; default is 1) specifies the how many mines per second the boss lays when laying mines. Currently, the boss lays mines two at time, so make sure `minesPerSec` is an interval of two.
- **`tractorTime`** (floating point; default is 1.0) specifies the time (in seconds) in which the boss uses tractor beams on the player.

Example:

```
AIDataValue ( "maxTractorDistance",  Float,    ( Val = 1500.0 ))
AIDataValue ( "minesPerSec",         Integer,  ( Val =    2 ))
AIDataValue ( "mineLayRange",        Float,    ( Val =  400.0 ))
AIDataValue ( "tractorTime",         Float,    ( Val =   12.0 ))
```

4.2.23.18 Blasting Out the Level

- *Klown Code File:* exc24.fsm

- *Klown:* DEvil_24_BlowOut
  —*Type:* Submachine
  —*Uses External Klowns:* BlastWave

```
// Boss-specific parameters
AIDataValue ( "blowOutHealth",       Integer,  ( Val = <#> ))
AIDataValue ( "blowOutSeq",          Resource, ( Val = ( rtSequence, <resource ID> )))

// Standard blast-wave parameters
AIDataValue ( "blastSensor",         Resource, ( Val = ( rtBodySensor, EXC24_BLAST_SENSOR
                ) ))
AIDataValue ( "blastDamage",         Float,    ( Val = 0 ))
AIDataValue ( "MaxForce",            Float,    ( Val = 100000000 ))
AIDataValue ( "BlastWidth",          Float,    ( Val = 2000 ))
AIDataValue ( "BlastWaveTime",       Float,    ( Val = EXC24_BLASTWAVE_TIME))
AIDataValue ( "blastSequence",       Resource, ( Val = ( rtSequence, EXC24_BLAST_SEQ )))
AIDataValue ( "MaxBlastRandomForce", Float,    ( Val = 900000 ))
AIDataValue ( "MaxBlastTorque",      Float,    ( Val = 9000000 ))
AIDataValue ( "MaxBlastShake",       Float,    ( Val = 0.2 ))
AIDataValue ( "blastChargeTime",     Float,    ( Val = 0.0 ))
```

When damage to the boss reaches a certain point, the boss emits a blast wave that clears everything out of the level except the boss and the player. The battle then becomes a *mano-y-sinistaro* conflict with no more powerups or crystals to help to player. To control this, set the following values:

- **blowOutHealth** (integer; default is 0) specifies the health below which the boss emits the level-clearing blast wave.
- **blowOutSeq** (resource; no default) specifies the `rtSequence` resources that plays on the boss when it blows out the level. Use this sequence to play all special effects involved in the blow out, such as changing the boss's model. Switch to the damaged model using an `rtModelChangeAction` action and set its physics using an `rtChangePhysicsAction` action. (The damaged model must have the exact same skeleton as the original boss.)

Example:

```
// Boss-specific parameters
AIDataValue ( "blowOutHealth",       Integer,  ( Val = 6 ))
AIDataValue ( "blowOutSeq",          Resource, ( Val = ( rtSequence, EXC24_SEQ_BLOWOUT )))

// Standard blast-wave parameters
AIDataValue ( "blastSensor",         Resource, ( Val = ( rtBodySensor, EXC24_BLAST_SENSOR
                ) ))
AIDataValue ( "blastDamage",         Float,    ( Val = 0 ))
AIDataValue ( "MaxForce",            Float,    ( Val = 100000000 ))
AIDataValue ( "BlastWidth",          Float,    ( Val = 2000 ))
AIDataValue ( "BlastWaveTime",       Float,    ( Val = EXC24_BLASTWAVE_TIME))
AIDataValue ( "blastSequence",       Resource, ( Val = ( rtSequence, EXC24_BLAST_SEQ )))
AIDataValue ( "MaxBlastRandomForce", Float,    ( Val = 900000 ))
AIDataValue ( "MaxBlastTorque",      Float,    ( Val = 9000000 ))
AIDataValue ( "MaxBlastShake",       Float,    ( Val = 0.2 ))
AIDataValue ( "blastChargeTime",     Float,    ( Val = 0.0 ))
```

In addition to the above two parameters, you must specify a blast wave for the boss, using the Blast Waves parameters. (A blast wave is already defined for `EXCESSION_24` in monster.r, so just adjust these parameters if necessary.) The blast wave does not affect the player (neither applying a force nor damage); it simply clears out the level. The wave's AI data values represent the force applied to objects and the amount of time it takes for the blast wave to travel through the level.

There are three blast-wave macros which you may want to tweak with are:

```
#define EXC24_BLASTWAVE_DISTANCE <#>
#define EXC24_BLASTWAVE_TIME     <#>
#define EXC24_BLASTWAVE_FADETIME <#>
```

These are defined right above the `rtAI` for `EXCESSION_24` in monster.r. These control the blast wave distance, the total time for the blast wave, and the amount of time over which the wave fades out once it is finished. The blast wave sequences and AI data values shown above use these `#define`s to produce the appropriate-looking effect.

Example:

```
#define EXC24_BLASTWAVE_DISTANCE 15000
#define EXC24_BLASTWAVE_TIME     7
#define EXC24_BLASTWAVE_FADETIME 1
```

5 Warriors and Transports

Sections 5.1-5.4 cover the four ship klowns present in warrior.fsm: warriors and transports. Section 5.5 covers the warrior behavior klown in warbeh.fsm.

5.1 Warrior Klown

- *Klown Code File:* warrior.fsm
- *Klown:* Warrior
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, WarriorPain, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, Weapons, FireControl, ElectronConductive, WarlikeBehavior, WarriorMissile, LaunchChild, Skeleton, Warrior_WavingSpikes, WarriorWings, Scorable, CloakedEnemy
  —*Has Internal Klown:* WarriorBehavior

The Warrior Klown manages warriors. Its internal WarriorBehavior Klown has no AI parameters of its own, although it does reference some parameters defined for the WarlikeBehavior Klown. The WarriorBehavior Klown oversees the following:

- If the warrior acquires a new target and is not assigned to stay in a fleet, it goes over to the Attack Banzai behavior.
- If the warrior's health changes and it is not assigned to stay in a fleet, it checks if it health is low enough to trigger a behavior change. It first checks for the Flee & Die behavior and then for thes Chase & Flee behavior.
- When a warrior stops fleeing, it looks for a target. If it finds one, it goes over to the Attack Banzai behavior. Otherwise, it assumes the Idle behavior.
- If a warrior's target dies, the warrior stops trying to orient toward the target.

5.2 HarrierWithBoosters Klown

- *Klown Code File:* warrior.fsm
- *Klown:* HarrierWithBoosters
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, WarriorPain, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, Weapons, FireControl, PollUpdate, ElectronConductive, Skeleton, Warrior_WavingSpikes, WarriorWings, Scorable, WarlikeBehavior, WarriorBoost, LaunchChild
  —*Has Internal Klown:* HarrierBehavior

The HarrierWithBoosters Klown manages warriors that have boosters, which they can use for bursts of speed and maneuverability. Its internal HarrierBehavior Klown has no AI parameters of its own, although it does reference some parameters defined for the WarlikeBehavior Klown. The HarrierBehavior Klown oversees the following:

- If the warrior acquires a new target and is not assigned to stay in a fleet, it turns on its boosters and goes over to the Attack Banzai behavior.

- If the warrior's health changes and it is not assigned to stay in a fleet, it checks if it health is low enough to trigger a behavior change. It first checks for the Flee & Die behavior and then for thes Chase & Flee behavior. If either behavior is triggered, the warrior turns on its boosters as it performs the behavior.
- The warrior periodically checks its condition. If it doesn't have a target, it turns off its boosters (if they are on). If it does have a target but is more than twice its `maxAttackDistance` from the target, it turns on its boosters (if they are off).
- When a warrior stops fleeing, it looks for a target. If it finds one, it turns on its boosters and goes over to the Attack Banzai behavior. Otherwise, it assumes the Idle behavior.
- If a warrior's target dies, the warrior stops trying to orient toward the target.

5.3 TransportWithTurrets Klown

- *Klown Code File:* warrior.fsm
- *Klown:* TransportWithTurrets
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, WarriorPain, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, Weapons, FireControl, ElectronConductive, TargetOffset, Skeleton, Warrior_WavingSpikes, WarriorWings, WarlikeBehavior, Attached, LaunchMother, LaunchChild, Scorable
  —*Has Internal Klown:* TransportBehavior

The TransportWithTurrets Klown manages warriors that have turrets; some of these also carry ships, which they can launch. Its internal TransportBehavior Klown has one AI parameter of its own, and it also references some parameters defined for the WarlikeBehavior Klown.

Note that in *Sinistar*, the TransportWithTurrets Klown manages the Sporg Guardian (TICK4) and Distilled Evil Executionery (DE_APC), both of which have turrets but do not carry ships, and the Sporg Attack Carrier (APC) and Sporg Battlecruiser (TROOP), both of which have turrets and carry ships.

5.3.1 TransportBehavior Klown Parameter

```
AIDataValue ( "launchDistance",   Float,     ( Val = <#>  ))
```

- **`launchDistance`** (floating point; no default) specifies the distance (in meters) from the warrior's target at which the warrior assumes its attack state and starts launching its ships.

Example:

```
AIDataValue ( "launchDistance",   Float,     ( Val = 1500.0 ))
```

5.3.2 TransportBehavior Klown Responsibities

The TransportBehavior Klown oversees the following:

- When a warrior with this klown has a target, it goes to its Attack Banzai behavior if it is not part of a fleet.
- A warrior with a target periodically checks its distance to the target. If the distance is less than or equal to its `launchDistance` (see 5.3.1 above), it triggers its Scramble message, which starts launching its on-board ships.

- If the warrior has a target when it launches a ship, it has the launched ship acquire that target. (This causes the launched ship to go attack the target.) Otherwise, the launched ship escorts the warrior.
- If the warrior's health changes and it is not assigned to stay in a fleet, it checks if it health is low enough to trigger a behavior change. It first checks for the Flee & Die behavior and then for thes Chase & Flee behavior.
- When a warrior stops fleeing, if it has a target, it goes over to the Attack Banzai behavior. Otherwise, it assumes the Idle behavior.
- If a warrior's target dies, the warrior stops trying to orient toward the target.

5.4 UnarmedTransport Klown

- *Klown Code File:* warrior.fsm
- *Klown:* UnarmedTransport
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, WarriorPain, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, Weapons, FireControl, ElectronConductive, TargetOffset, Skeleton, Warrior_WavingSpikes, WarriorWings, WarlikeBehavior, Scorable, Team, DeathPowerup, WarlikePatrolling
  —*Has Internal Klown:* UnarmedTransportBehavior

The UnarmedTransport Klown manages the Sporg and Distilled Evil transports (LITETRANS and EVIL_TRANSPORT), which carry power-ups. Its internal UnarmedTransportBehavior Klown has no AI parameters of its own, although it references some parameters defined for the WarlikeBehavior Klown. The UnarmedTransportBehavior Klown oversees the following:

- If the transport's health changes and it is not assigned to stay in a fleet, it checks if it health is low enough to trigger a behavior change. It first checks for the Flee & Die behavior and then for thes Chase & Flee behavior.

5.5 Warrior Behaviors

- *Klown Code File:* warbeh.fsm
- *Klown:* WarlikeBehavior
  —*Type:* Submachine
  —*Uses External Klowns:* PollUpdate, PollFast, SetTarget, BarrelRollEngines, MatchEngines, MoveEngines, OrientEngines, CircleEngines

5.5.1 Respawning

```
AIDataValue ( "respawnWarrior",    Integer,    ( Val = <flag> )).
AIDataValue ( "respawnId",         Integer,    ( Val = <container's resource ID> )
AIDataValue ( "respawnDistance",   Float,      ( Val = <#>  ))
```

Once the warriors die they can be reincarnated into any other form. This is very similar to the workers' respawning ability. The parameters that govern respawning are:

- **respawnWarrior** (integer; 0 or 1; default is 0) specifies whether the current warrior should respawn (`Val = 1`) or not (`Val = 0`).
- **respawnId** (resource ID; default is 1) specifies the resource ID of the container (object) that is respawned. Note that this respawned object can be any container available for use on the level and not just the same

container as the warrior that was destroyed. For example, a `TICK1` warrior could be respawned as a `FIGHTER` heavy fighter or a `TICK2` worker or so on.

- **respawnDistance** (floating point; default is 1.0) specifies the distance (in meters) from the center of the universe that the respawned object appears. It appears at a random position anywhere on the sphere specified by the `respawnDistance`.

Note: `respawnId` and `respawnDistance` actually come from the MiningManager klown (used for workers), but the MissionControl klown has press-ganged them for use with warriors, too.

Example:

```
AIDataValue ( "respawnWarrior",    Integer,    ( Val = 1 ))
AIDataValue ( "respawnId",         Integer,    ( Val = FIGHTER ))
AIDataValue ( "respawnDistance",   Float,      ( Val = 100.0 ))
```

In this example the destroyed ship will reincarnate at a distance of 100 meters from center of the universe as a Sporg **heavy fighter.**

5.5.2 Roll Away

```
AIDataValue ( "rollAwayDistance",   Float,    ( Val = <#> ))
AIDataValue ( "rollAwayRadius",     Float,    ( Val = <#> ))
```

Roll Away behavior gives a warrior a special ship-attacking tactic in which it starts attacking and then suddenly rolls over at close range and flies away, firing on the player whenever possible. Roll Away behavior requires two parameters:

- **rollAwayDistance** (floating point; default is 0.0) specifies the distance (in meters) from the player at which the warrior starts rolling away from the player.
- **rollAwayRadius** (floating point; default is 100.0) specifies the radius (in meters) of the roll. The roll consists of the warrior moving 180 degrees along the circle described by the roll away distance and the roll away radius, so note that the warrior rolls through the diameter of the circle (i.e., twice the value for the roll away radius). Once the roll is over, the warrior starts moving normally again.

A warrior moves at its maximum speed when rolling away. The specific direction in which the warrior rolls depends upon its up vector when it starts the roll: it always rolls in the direction of its up vector.

Example:

```
AIDataValue ( "rollAwayDistance",   Float,    ( Val =  350.0 ))
AIDataValue ( "rollAwayRadius",     Float,    ( Val = 1000.0 ))
```

5.5.3 Swoop Past

```
AIDataValue ( "swoopDistance",   Float,    ( Val = <#> ))
```

Swoop Past behavior is a hack-and-slash kind of behavior in which the warrior heads for the player at full speed as if it is ramming and then suddenly veers away from the player. This can cause the player to maneuver to avoid the warrior's seeming suicidal tendencies. This behavior can be useful for ships that are fast but sluggish. It requires just one parameter:

- **swoopDistance** (floating point; default is 0.0) specifies the distance (in meters) from player at which the warrior suddenly changes its course.

Example:

```
AIDataValue ( "swoopDistance",   Float,   ( Val = 500.0 ))
```

5.5.4 Barrel Roll

```
AIDataValue ( "barrelRollDistance",   Float,   ( Val = <#> ))
```

Warriors that use this behavior make it harder for the player to shoot and kill them. When using this tactic, a warrior attaching the player barrel rolls in circular motion until it is in close proximity to player. A barrel roll by a warrior uses the BarrelRoll common ship behavior (`barrelSpinEffort`, `barrelStrafeEffort`) as well as the following parameter:

- **`barrelRollDistance`** (floating point; default is 100.0) specifies the distance (in meters) from the player at which the warrior begins to barrel roll.

Example:

```
AIDataValue ( "barrelRollDistance",   Float,   ( Val = 1300.0 ))
AIDataValue ( "barrelSpinEffort",     Float,   ( Val =    1.0 ))
AIDataValue ( "barrelStrafeEffort",   Float,   ( Val =    0.0 ))
```

5.5.5 Escort Service

```
AIDataValue ( "escortObject",   Integer,   ( Val = <master's resource ID> ))
AIDataValue ( "escortRadius",   Float,     ( Val = <#> ))
```

A warrior using this behavior escorts and guards its specified master, which can be any object like another ship, a Sinistar, the jumpgate, etc. A warrior on escort service attacks only when it detects danger to its master: when a player is within the standard `maxAttackDistance` of the warrior, the warrior attacks the player. After scaring away the player, the warrior always comes back to protect its master. The distance at which the warrior decides to return to its master again depends on the standard `maxAttackDistance` of the warrior: if the player is beyond the distance specified in this parameter, the warrior returns to escort its master. The Escort Service behavior itself takes two parameters:

- **`escortObject`** (resource ID; no default) specifies the resource ID of the container that is the master for the escorting warrior.
- **`escortRadius`** (floating point; default is 1000.0) specifies the radius (in meters) of the sphere around the master in which the warrior moves while escorting the master.

If the master gets killed, a warrior escorting it is released from escort service and subsequently acts like any other standard warrior of its class.

Example:

```
AIDataValue ( "escortObject",   Integer,   ( Val = APC ))
AIDataValue ( "escortRadius",   Float,     ( Val = 500.0 ))
```

5.5.6 Chase & Flee

```
AIDataValue ( "chaseHealth",     Integer,   ( Val = <#> ))
AIDataValue ( "chaseRadius",     Float,     ( Val = <#> ))
AIDataValue ( "chaseDistance",   Float,     ( Val = <#> ))
AIDataValue ( "fleeAngle",       Float,     ( Val = <#> ))
```

```
AIDataValue ( "fleeAvoidAngle",  Float,     ( Val = <#> ))
```

When a warrior takes sufficient damage, it tries to save itself by escaping from the player, fleeing and dodging shots if the player chases it. The warrior will not flee beyond a certain distance and will go back over to the attack if the player stops chasing it. (If the warrior returns to the attack and subsequently takes damage, Chase & Flee behavior can be triggered again.) Three parameters govern this behavior:

- **chaseHealth** (integer; no default) is the warrior's health value at which or below the Chase & Flee behavior is triggered
- **chaseRadius** (floating point; default is 500.0) is the maximum distance (in meters) from the center of the universe beyond which the fleeing warrior will not go. The warrior will always move to stay within this distance.
- **chaseDistance** (floating point; default is 500.0) is the maximum distance (in meters) the warrior uses to determine if the player is chasing it. If the player is beyond this distance, the warrior decides it is no longer being chased, stops, fleeing, and resumes trying to attack the player.
- **fleeAngle** (floating point; default is 0.8) is the cosine of the angle that the warrior turns when fleeing the player. (An random component makes it equally likely that it will turn clockwise or counterclockwise when using this angle.)
- **fleeAvoidAngle** (floating point; default is 0.9) is the cosine of the angle that defines the player's "cone of influence" used for fleeing. (A different cone is used for barrel rolls.) This cone extends from the player ship in the direction the ship is facing, and is presumed to be the volume of space in which the player is most likely to shoot at and hit the warrior. Accordingly, the warrior tries to stay outside this cone when fleeing.

Notes: 1) Chase & Flee behavior is different from Flee & Die behavior. These two behaviors work best if `chaseHealth` is set to a larger value than `fleeHealth`. Flee & Die behavior always takes precedence over Chase & Flee behavior. 2) `fleeAngle` and `fleeAvoidAngle` are use for both Chase & Flee and Flee & Die behaviors.

Example:

```
AIDataValue ( "chaseHealth",     Integer,   ( Val =   54.0 ))
AIDataValue ( "chaseDistance",   Float,     ( Val = 1000.0 ))
AIDataValue ( "chaseRadius",     Float,     ( Val = 2000.0 ))
AIDataValue ( "fleeAngle",       Float,     ( Val =    0.5 ))
AIDataValue ( "fleeAvoidAngle",  Float,     ( Val =    0.8 ))
```

5.5.7 Hide in Roid

```
AIDataValue ( "likesRoid",    Integer,    ( Val = <flag> ))
AIDataValue ( "roidSensor",   Resource,   ( Val = ( rtBodySensor, <resource ID> ))
```

Hide in Roid behavior is similar to Chase & Flee behavior, but while escaping from the player the warrior likes to hide among asteroids. To use Hide in Roid behavior, specify the chase and flee parameters per above and also use this parameter:

- **likesRoid** (integer; 0 or 1; default is 0) specifies whether the warrior likes to hide among the roids (`Val = 1`) or not (`Val = 0`, the default) when fleeing.
- **roidSensor** (resource; no default) specifies the sensor the warrior uses to detect asteroids—a warrior that likes to hang with the roids should have a roid sensor! This resource must be defined separately. See Sensors for more details on sensors.

Example:

```
AIDataValue ( "likesRoid",    Integer,    ( Val = 1 ))
AIDataValue ( "roidSensor",   Resource,   ( Val = ( rtBodySensor, TICK_ROID_SENSOR ))
```

5.5.8 Fly Away

```
AIDataValue ( "flyAwayDistance",   Float,   ( Val = <#> ))
AIDataValue ( "waitTime",          Float,   ( Val = <#> ))
```

All the above behaviors that involve attacking the player also have a Fly Away behavior. After a warrior has attacked the player, it will fly away from player and then return to make another attack using its attack behavior. Fly Away behavior uses:

- **flyAwayDistance** (floating point; default is 400.0) is the distance (in meters) that the warrior flies away from the player (the player's position at the instant the warrior's attack ends) before returning to attack again. (When the warrior begins its Fly Away behavior, it selects a position at the `flyAwayDistance` and moves to that position.)
- **waitTime** (floating point; default is 1.0) controls what happens when a warrior using the Fly Away behavior cannot reach its `flyAwayDistance` position due to the avoidance system (i.e., something or things are blocking it). `waitTime` therefore specifies the time (in half seconds) that the warrior waits trying to reach its `flyAwayDistance` position. If it does not reach its `flyAwayDistance` position within its `waitTime`, it ends its Fly Away behavior without reaching its `flyAwayDistance` position.

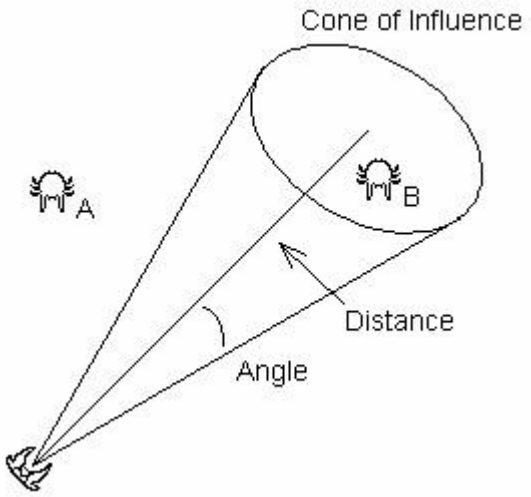
Example:

```
AIDataValue ( "flyAwayDistance",   Float,   ( Val = 1900.0 ))
AIDataValue ( "waitTime",          Float,   ( Val =    3.5 ))
```

5.5.9 Shot Deflector

```
AIDataValue ( "barrelEscape",           Integer,   ( Val = <flag> ))
AIDataValue ( "barrelEscapeDistance",   Float,     ( Val = <#> ))
AIDataValue ( "barrelEscapeAngle",      Float,     ( Val = <#> ))
```

Shot Deflector behavior causes a warrior to check to see if it is in a position likely to be shot at by the player and, if so, to barrel roll (see Barrel Roll) to avoid being fired upon. As shown in the diagram, the shot deflector klown defines a cone of influence based on the player, which is assumed to be the most like volume of space in which the player can fire and hit a target. Thus, warriors using Shot Deflection behavior will use this behavior if they are within the cone of influence (in the diagram, Warrior B will do this) but will not if they are outside the cone (Warrior A). For this behavior to work, the barrel roll AI data values must also be specified in addition to these shot deflector parameters:



- **barrelEscape** (integer; 0 or 1; default is 0) flags whether (`Val = 1`) or not (`Val = 0`, the default) the warrior uses the Shot Deflector behavior.
- **barrelEscapeDistance** (floating point; => 0; default is 500.0) specifies distance (in meters) from the player within which the warrior will use Shot Deflector behavior, if the warrior is in the player's cone of influence.

- **barrelEscapeAngle** (floating point; default is 0.9, approximately 26 degrees) specifies the cosine of the angle (see Quick Table of Cosines) used to determine the player's cone of influence (see diagram).

Example:

```
AIDataValue ( "barrelEscape",           Integer,   ( Val = 1 ))
AIDataValue ( "barrelEscapeDistance",   Float,     ( Val = 1100.0 ))
AIDataValue ( "barrelEscapeAngle",      Float,     ( Val =    0.9848 ))
```

5.5.10 Flee & Die

```
AIDataValue ( "fleeHealth",      Integer,   ( Val = <#> ))
AIDataValue ( "fleeDistance",    Float,     ( Val = <#> ))
AIDataValue ( "fleeAngle",       Float,     ( Val = <#> ))
AIDataValue ( "fleeAvoidAngle",  Float,     ( Val = <#> ))
```

When a warrior takes sufficient damage, it tries to save itself by escaping from the player, fleeing and dodging shots if the player chases it. Once a warrior starts to flee, it does not stop fleeing and will eventually kill itself (thereby allowing, for example, Respawning behavior to kick in). Two parameters govern this behavior:

- **fleeHealth** (integer; default is -1) is the warrior's health value at which or below the Flee & Die behavior is triggered.
- **fleeDistance** (floating point; >= 0.0; default is 9000.0) is the distance (in meters) from the center of the universe at which the fleeing warrior kills itself.
- **fleeAngle** (floating point; default is 0.8) is the cosine of the angle that the warrior turns when fleeing the player. (An random component makes it equally likely that it will turn clockwise or counterclockwise when using this angle.)
- **fleeAvoidAngle** (floating point; default is 0.9) is the cosine of the angle that defines the player's "cone of influence" used for fleeing. (A different cone is used for barrel rolls.) This cone extends from the player ship in the direction the ship is facing, and is presumed to be the volume of space in which the player is most likely to shoot at and hit the warrior. Accordingly, the warrior tries to stay outside this cone when fleeing.

Notes: 1) Flee & Die behavior is different from Chase & Flee behavior. These two behaviors work best if `chaseHealth` is set to a larger value than `fleeHealth`. Flee & Die behavior always takes precedence over Chase & Flee behavior. 2) `fleeAngle` and `fleeAvoidAngle` are use for both Chase & Flee and Flee & Die behaviors.

Example:

```
AIDataValue ( "fleeHealth",      Integer,   ( Val =     2.0 ))
AIDataValue ( "fleeDistance",    Float,     ( Val = 10000.0 ))
AIDataValue ( "fleeAngle",       Float,     ( Val =     0.5 ))
AIDataValue ( "fleeAvoidAngle",  Float,     ( Val =     0.8 ))
```

5.5.11 Attack Distances

```
AIDataValue ( "maxAttackDistance",   Float, ( Val = <#> ))
AIDataValue ( "minAttackDistance",   Float, ( Val = <#> ))
```

Warriors have attack distance parameters:

- **maxAttackDistance** (floating point; no default) specifies the maximum attack distance of the warrior.

- **minAttackDistance** (floating point; no default) specifies the minimum attack distance of the ship.

If a warrior is further than its maximum attack distance from its target (typically, the player ship), it attempts to close on the target until it is within its minimum attack distance.

Note: Bosses also have `maxAttackDistance` and `minAttackDistance` parameters, but they work somewhat differently than these warrior parameters.

Example:

```
AIDataValue ( "maxAttackDistance",   Float, ( Val = 1500.0 ))
AIDataValue ( "minAttackDistance",   Float, ( Val =  600.0 ))
```

5.5.12 Evade & Follow

```
AIDataValue ( "followDistance",   Float,   ( Val = <#> ))
```

If a warrior comes within its `followDistance` of the player ship, it tries to evade the player ship and get behind it. For up to the next 5.0 seconds (hardcoded), the warrior maneuvers trying to get behind the player ship. If it succeeds, it starts firing on the player ship. Otherwise, at the end of the five seconds, it abandons its Evade & Follow behavior and moves toward the player ship (without necessarily trying to get behind the player ship). The following parameter triggers this behavior:

- **followDistance** (floating point; default is -1) specifies the distance to the player ship that triggers the Evade & Follow behavior. If `followDistance` is not specified, the default value means the Evade & Follow behavior cannot be triggered. *Note:* The klown also calculates a minimum distance value, which is three times the sum of the warrior's radius and the player ship's radius. If this minimum value is greater than the value set in `followDistance`, then the minimum value is used instead.

Example:

```
AIDataValue ( "followDistance",   Float,   ( Val = 200.0 ))
```

5.5.13 Move to Waypoints

```
AIDataValue ( "waypoint",        Vector,   ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "waypointRange",   Float,    ( Val = <#> ))
```

A warrior can move to a specified waypoint. When it comes within range of the waypoint, it ceases moving to the waypoint. (It sends a message to mission control when it arrives at a waypoint in case getting to the waypoint is victory condition.)

Warriors uses waypoints while using WarlikePatrolling and when they don't have targets while using WarlikeBehavior.

The following parameter control Move to Waypoint behavior:

- **waypoint** (vector; default is 0.0, 0.0, 0.0) specifies a waypoint for the warrior. If the default is used, the warrior does not use Move to Waypoints behavior.
- **waypointRange** (floating point; default is twice the warrior's radius) specifies the distance (in meters) to the waypoint at which the warrior stops moving to the waypoint.

Note: The WarlikePatrolling klown also contains code for having a queue of several waypoints, but it is not apparently as of this writing how the queue gets specified. (A parameter is to be added later?)

Example:

```
AIDataValue ( "waypoint",        Vector,   ( Val = ( 100.0, 200.0, 100.0 )))
AIDataValue ( "waypointRange",   Float,    ( Val = 50.0 ))
```

6 Workers

- *Klown Code File:* worker.fsm
- *Klown:* Worker
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, Slave, MoveEngines, OrientEngines, PollUpdate, PollFast, PollVeryFast, ElectronConductive, Team, Avoidance, Scorable, CircleEngines
  —*Has Internal Klown:* NormalBehavior

Workers search for and mine crystals, carting them off to the jumpgate. Workers are unarmed and mine asteroids by colliding with them. Workers can also land at asteroids equipped with landing pads. The Worker klown controls all worker behaviors:

- A worker ignores damage from collisions with a jumpgate.
- A worker asks Mission Control for intructions as needed and follows those instructions.
- Under typical conditions, a worker without a crystal looks for a roid to mine, using its roid sensor. It selects the nearest roid that has 1 or more crystals (it ignores crystal-less roids) and is not moving too fast. (If the worker tried to go to a roid but failed to get there after a while, the roid is tagged as being moving too fast.)
- After mining a roid, the worker will use its crystal sensor to search for a crystal to grab. It ignores crystals that are moving too fast for it to catch and crystals that have morphed (to energize a jumpgate). If it finds a crystal, it checks with Mission Control to see if it should grab it (Mission Control may have assigned it to another worker).
  If it get the green light, it goes after the crystal, turning off its avoidance ability. If the crystal disappears (such as the player ship eating it or another worker managing to grab it despite Mission Control), the worker gives up on it. It also gives up if it can't catch up to if after a period of time. If everything works right, the worker grabs the crystal, using its grabbing articulation sequence, and turns its avoidance ability back on.
- If a worker is trying to chase down a crystal and bumps into another crystal, it will try to grab that crystal instead.
- Once a worker grabs a crystal, it carts it off to a position specified by its master and delivers it there, using its releasing articulation sequence. If its delivery position get filled before it can deliver the crystal, it asks its master for a new delivery position.
- If the worker has a crystal but its master no longer needs any, the worker goes into deep space and dumps it. If the master changes its mind before the worker dumps it (such as due to the player destroying crystals around the jumpgate), the worker obediently turns back to deliver it.
- The worker has various other behaviors, such as Idle (where it flies to a position near its master to search for roids), Move to Home (where it flies to a point near its pre-set home position), Full Stop (where it hangs out at its home position doing nothing until it is reactivated), and Circle Home Position.
- Mission Control can tell workers to go chase the player. (This was for a game feature not used in *Sinistar*.) When told to chase the player, the worker looks for the player (using its player sensor). When found, the worker checks with Mission Control to see it if still is OK, and hies off after the player when allowed. It will continue to chase the player until 1) Mission Control tells it to stop, 2) the player dies, 3) the player cloaks, 4) the player changes teams, or 5) the distance to the player is greater than the worker's `maxChaseDistance`.
- If Mission Control no longer wants a worker around, it can tell it to self destruct. The worker's health is set to -1, which causes the worker to be destroyed.

6.1.1 Basic Worker Parameters

```
AIDataValue ( "miningDamage",        Float,   ( Val = <#> ))
AIDataValue ( "roidLandDistance",    Float,   ( Val = <#> ))
AIDataValue ( "maxChaseDistance",    Float,   ( Val = <#> ))
AIDataValue ( "deliverDistance",     Float,   ( Val = <#> ))
AIDataValue ( "dumpDistance",        Float,   ( Val = <#> ))
```

Worker behaviors are controlled by these parameters:

- **miningDamage** (floating point; default is 0.0) specifies the amount of damage the worker does to an asteroid when it collides with one. (This helps the worker mine asteroids.)
- **roidLandDistance** (floating point; no default) specifies the distance (in meters) from a roid with a landing pad that the worker must be to land at that roid.
- **maxChaseDistance** (floating point; no default) specifies the distance (in meters) from the player that worker chasing the player will give up the chase.
- **deliverDistance** (floating point; default is 1.0) specifies the distance (in meters) from a specified position that the worker delivers a crystal it is carrying. This position is set by the master to which the worker is enslaved. Typically, the master is a jumpgate and directs workers to bring crystals to positions around the gate.
- **dumpDistance** (floating point; no default) specifies the distance (in meters) into deep space that a worker goes to dump a crystal that is no longer needed by its master.

Example:

```
AIDataValue ( "miningDamage",        Float,   ( Val =    5.0 ))
AIDataValue ( "roidLandDistance",    Float,   ( Val =   20.0 ))
AIDataValue ( "maxChaseDistance",    Float,   ( Val =  500.0 ))
AIDataValue ( "deliverDistance",     Float,   ( Val =   20.0 ))
AIDataValue ( "dumpDistance",        Float,   ( Val = 2000.0 ))
```

6.1.2 Worker Articulations

```
AIDataValue ( "crystal_grab",          Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "crystal_release",       Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "crystalStoragePosition", Vector,  ( Val = <vector> ))
AIDataValue ( "crystalMoveTime",        Float,   ( Val = <#> ))
```

These parameters describe the articulation of workers' arms as the workers grab, carry, and release crystals. Workers grab crystals in space using their mouths, move the crystals to their abdomens for secure transport, and then carry them to the vicinity of the jumpgate, where they then release the crystals. There are four parameters to be specified:

- **crystal_grab** (resource; no default) specifies the resource ID of the grabbing sequence that is played when the worker grabs a crystal.
- **crystal_release** (resource; no default) specifies the resource ID of the release sequence that is played when the worker releases a crystal it is carrying.
- **crystalStoragePosition** (vector; default is 0.0, 0.0, 15.0specifies the X-Y-Z position (in the worker's local coordinates) relative to the worker's center where the crystal will be held while the worker carries it to the gate.

- **crystalMoveTime** (floating point; default is 0.5) specifies the time (in seconds) it takes the worker to move a crystal from its mouth to its abdomen.

Example:

```
AIDataValue ( "crystal_grab",           Resource, ( Val = ( rtSequence,
                SEQ_TICK2B_CRYSTAL_GRAB )))
AIDataValue ( "crystal_release",        Resource, ( Val = ( rtSequence,
                SEQ_TICK2B_CRYSTAL_RELEASE )))
AIDataValue ( "crystalStoragePosition", Vector,   ( Val = ( 0.0, 0.0, 15.0 )))
AIDataValue ( "crystalMoveTime",        Float,    ( Val = 0.75 ))
```

6.1.3 Worker Sensors

Workers should have the following sensors defined:

- **crystalSensor** (resource; default is no sensor) specifies the sensor the worker uses to detect crystals.
- **playerSensor** (resource; default is no sensor) specifies the sensor the worker uses to detect the player.
- **roidSensor** (resource; default is no sensor) specifies the sensor the worker uses to detect asteroids.

See Sensors for details on sensors.

7 Special Ships

7.1 The Harvester (Harvester Klown)

- *Klown Code File:* harvester.fsm
- *Klown:* Harvester
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, MoveEngines, OrientEngines, Avoidance, SetTarget, SearchForTarget, ElectronConductive, Skeleton, Warrior_WavingSpikes, WarriorPain, Attached, TractorMine, WarlikeBehavior, Scorable
  —*Has Internal Klown:* HarvesterBehavior

```
AIDataValue ( "safeDistance",        Float,    ( Val = <#> ))
AIDataValue ( "miningDamage",        Float,    ( Val = <#> ))
AIDataValue ( "maxCrystalPower",     Float,    ( Val = <#> ))
AIDataValue ( "startCrystalPower",   Float,    ( Val = <#> ))
```

The Harvester is a massive, armed worker that appears on Level 4. It tractors roids into its mouth and can repel missiles and the player with the tractor beam. It can attack the player with its weapons. (At an early point in the design, the Harvester was intended to give shields to warriors and to even create a warrior if it mined enough, but these features was not included in the game.) Some of the Harvester's behaviors are controlled by tunable parameters:

- **safeDistance** (floating point; default is 2000.0) specifies the distance (in meters) beyond which the Harvester feels safe from the player.
- **miningDamage** (floating point; default is 0.0) specifies the amount of damage the Harvester does to an asteroid when it collides with one. (This helps the Harvester mine asteroids.)
- **maxCrystalPower** (floating point; default is 0.0) specifies the maximum power the Harverster can gain from crystals.
- **startCrystalPower** (floating point; default is 90% `maxCrystalPower`) specifies the starting power the Harverster has. If `startCrystalPower` is not defined, then the Harvester starts with 90% of its `maxCrystalPower`.

Example:

```
AIDataValue ( "safeDistance",        Float,    ( Val = 2000.0 ))
AIDataValue ( "miningDamage",        Float,    ( Val =   10.0 ))
AIDataValue ( "maxCrystalPower",     Float,    ( Val =  100.0 ))
AIDataValue ( "startCrystalPower",   Float,    ( Val =    0.0 ))
```

7.1.1 Power & Crystal Management

While the Harvester has crystals but has less than maximum power, it burns crystals, gaining power at a rate of 1 power unit per crystal.

If the Harvester ever has a negative number of crystals but has some power, it expends power, gaining crystals at a rate of 1 crystal per power unit.

7.1.2 Behavior Summary

<table>
<tr><td>Mine</td><td colspan="2">The Harvester starts the level mining asteroids. It mines ’roids until its Attack behavior is triggered by:</td></tr>
<tr><td rowspan="3"></td><td rowspan="3"></td><td>It reaching maximum power.</td></tr>
<tr><td>A timer (controlled by the level) telling it to attack the player.</td></tr>
<tr><td>The player coming within the Harvester’s detection range.</td></tr>
<tr><td>Attack</td><td colspan="2">The Harvester attacks the player using the Attack Banzai behavior of warriors.</td></tr>
<tr><td>Flee</td><td colspan="2">The Harvester’s flight behavior is triggered using the <code>fleeHealth</code> parameter from warrior behaviors (see Flee & Die). Once triggered, the Harvester flees from the player until it is at least its <code>safeDistance</code> from the player, whereupon it resumes its normal activities.</td></tr>
</table>

8 Space Stations

Levels 8 has the Sporg Base Station and 16 has the Distilled Evil Battle Station. Each station is composed of several modules. Each type of module has its own klown.

8.1 Launch Bay Module (StationLaunchBay Klown)

- *Klown Code File:* station.fsm
- *Klown:* StationLaunchBay
  —*Type:* CHFSM
  —*Uses External Klowns:* BossOnRadar, Birth, Health, SetTarget, SearchForTarget, Team, Attached, LaunchMother, Scorable
  —*Has Internal Klown:* StationLaunchBehavior

```
AIDataValue ( "openpanelSequence",   Resource,   ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "openAt",              Integer,    ( Val = <#> ))
```

A launch platform contain launch bays that can launch ships, using the standard `rtLaunchInfo` launching functionality. After the platform is initialized, it enters a state of Yellow Alert, in which it does not launch ships. It enters a state of Red Alert, opening its panels and launching ships, when it receives a Red Alert message (from a communications platform) or when it takes damage (presumably caused by an attacking player).

The klown controlling the launch platform has two parameters:

- **`openpanelSequence`** (resource, no default) specifies the sequence that plays when the platform opens its panels.
- **`openAt`** (integer; no default) specifies the time (in seconds) the platform waits before opening its flaps when going to Red Alert. *Warning:* The .r file notes "the game crashes for values of 1 or less"!

Example:

```
AIDataValue( "openpanelSequence", Resource, ( Val = ( rtSequence,
                LARGESAT2_OPENLAUNCHBAY_SEQ )))
AIDataValue( "openAt",            Integer,  ( Val = 3 ))
```

8.2 Defense Module (StationDefense Klown

- *Klown Code File:* station.fsm
- *Klown:* StationDefense
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, SetTarget, SearchForTarget, Team, Attached, Scorable (ElectronConductive klown present but commented out)
  —*Has Internal Klown:* StationDefenseBehavior

Note: The text below describes the StationDefense klown. However, it was not used in *Sinistar*, as the StationKey klown was used instead.

A defense platform starts at a state of Yellow Alert, with all of its type 0 turrets inactive, since they cannot shoot through the shield that is active. The platform enters a state of Red Alert, activating its type 0 turrets, when it receives a Red Alert message (from a communications platform) or when it takes damage (presumably caused by an attacking player).

The StationDefense klown has no custom parameters.

8.3 Communications Module (StationCom Klown)

- *Klown Code File:* station.fsm
- *Klown:* StationCom
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, SetTarget, SearchForTarget, Team, Scorable, Attached (ElectronConductive klown present but commented out)
  —*Has Internal Klown:* StationComBehavior

```
AIDataValue ( "haildelay",          Float,    ( Val = 60000 ))
AIDataValue ( "alertdelay",         Float,    ( Val = 30   ))
AIDataValue ( "StationComSequence", Resource, ( Val = ( rtSequence, LARGESAT2_COM )))
```

Note: The text below describes the StationDefense klown. However, it was not used in *Sinistar*, as the StationGenerator klown was used instead.

A communications platform starts at a state of Yellow Alert. The platform enters a state of Red Alert, signaling for help, after a set amount of time or after it takes damage (presumably caused by an attacking player). The concept is that the communications platform calls for help quickly if the player attacks the station. If the player does not attack, after a length of time, the satellite calls for help anyway. The player can destroy the communications platform before it sends out its call. The communications platform parameters are:

- **`haildelay`** (floating point; no default) specifies the time in ***milliseconds*** that must pass before the satellite calls in warriors, if the player does not fire on the satellite at all. ***Special Note:*** The info I received indicated that this value is in milliseconds, even though almost all other floating-point time parameters are in seconds.
- **`alertdelay`** (floating point; no default) specifies the time in ***seconds*** that must pass before the satellite calls in warriors, once the player starts to fire on the satellite. If the communications array is destroyed before this time passes, no warriors are called in.
- **`StationComSequence`** (resource; no default) specifies the sequence that plays when the communications platform signals for help. (This sequence can cause the relieving force ships to be emitted.)

8.4 Generator Module (StationGenerator Klown)

- *Klown Code File:* station.fsm
- *Klown:* StationGenerator
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Team, Attached, Scorable

The StationGenerator klown exists so that shield generators can be attached to the stations. The klown just has a set of external klowns and has no other functionality.

8.5 Key Module (StationKey Klown)

- *Klown Code File:* station.fsm
- *Klown:* StationKey
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Team, Scorable, Attached (ElectronConductive, Skeleton, TargetOffset, Selective Damage, SetTarget, SearchForTarget klowns present but commented out)
  —*Has Internal Klown:* StationKeyBehavior

The StationKey klown tells Mission Control that its container is the object that must be destroyed in the mission. Only one container can have this klown on a level.

Why not use the standard mission victory functionality rather than this klown? The reason is that the container does not exist at the time that MissionControl assigns the DeathCallBack message.

9 Egg Klown

- *Klown Code File:* egg.fsm
- *Klown:* Egg
  —*Type:* CHFSM
  —*Uses External Klowns:* Health, Collision, ElectronConductive, Team, MatchEngines
  —*Has Internal Klown:* EggGuts

```
AIDataValue ( "eggHatchTime",         Float,     ( Val = <#> ))
AIDataValue ( "eggHatchSequence",     Resource, ( Val = ( rtSequence, <resource ID> )))
AIDataValye( "defaultEggContainer", Integer,  ( Val = <resource ID> ))
AIDataValue ( "team",                 Integer,  ( Val = <flag> ))
```

Eggs can hatch objects like workers, warriors, and the Sinistar. For objects that use team identifiers, the egg sets the team of the hatchling to be the same as the team of the egg. Eggs are controlled by the following AI parameters:

- **eggHatchTime** is the amount of time (in seconds) until the egg hatches.
- **eggHatchSequence** is the sequence used to emit the child. The `rtSequence` resource for the `eggHatchSequence` sequence must have an `rtEmitInfo` action named "`eggEmitter`". This action is the one that hatches the child.
- **defaultEggContainer** (integer) specifies the resource ID of the default egg container to use for the egg. *Note:* This parameter is defined in the klown but is not used in the .r files in the game.
- **team** is the standard AI parameters that defines the team of the egg. However, note that ***the Egg klown sets the team of the child to match the team of the egg.*** This allows an egg to hatch for its own team an object specified for another team in the object's .r file. `EXCESSION_24`'s eggs use this ability, for example, to hatch Tick-team bosses as Distilled Evil-team bosses.

Notes: 1) When an egg hatches, its "observer" is notified that the egg has hatched. The observer typically is the object that laid the egg but in theory any object can be assigned observer status. Note that `EXCESSION_24` uses this feature to let its hatchlings fight the player while the boss itself hangs back. 2) Eggs laid by bosses have "brakes." When an egg is shot or rammed into, it will slow down and stop.

Example:

```
AIDataValue ( "eggHatchTime",        Float,     ( Val = 20.0 ))
AIDataValue ( "eggHatchSequence",    Resource, ( Val = ( rtSequence, SEQ_EGG_29_HATCH )))
AIDataValue ( "team",                Integer,  ( Val = TEAM_EVIL ))
)

defres rtSequence ( SEQ_EGG_29_HATCH )
(
   Actions[] =
   (
      ActionId = ( rtEmitInfo, ACT_EGG_29_HATCH )
      Name     = "eggEmitter"
      Time     = 0
   )
   ...
)

defres rtEmitInfo ( ACT_EGG_29_HATCH )
```

```
(
   NumParticles       = 1
   EmitTime           = 0.0
   Emitter.EmitSingle = ( ContainerId = EVIL_WARRIOR4 )
   INIT_POS_POINT_ORIGIN()
   INIT_ORIENT_FIXED_IDENTITY()
   INIT_VEL_FIXED( 0, 0, 0 )
)
```

10 ExcEgg Klown: Jumpgates

- *Klown Code File:* excegg.fsm
- *Klown:* ExcEgg
  —*Type:* HFSM
  —*Uses External Klowns:* PollUpdate, Collision, Master, BossOnRadar, Team, TractorBeam2, TargetOffset, ForceField, GateLightning

10.1 Mandatory Boss Arrival Time

```
AIDataValue ( "MaxBirthTime",       Float,    ( Val = <#> ))
```

Use the following parameter to set a mandatory time for the boss's arrival on a level:

- **MaxBirthTime** (floating point; default is 1000.0) is the time (in seconds) from the beginning of play on the level when the boss arrives on the level, whether or not the gate has sufficient crystals.

Example:

```
AIDataValue ( "MaxBirthTime",       Float,    ( Val = 1200.0 ))
```

10.2 Arrival Delay Time

```
AIDataValue ( "birthDelay",         Float,    ( Val = <#> ))
```

- **birthDelay** (floating point; default is 10.0) is the time (in seconds) the boss's arrival sequence is delayed once the gate is full of crystals.

Example:

```
AIDataValue ( "birthDelay",         Float,    ( Val = 2.0 ))
```

10.3 Boss Health & Gate Damage

```
AIDataValue ( "BossHealthChart", Resource, ( Val = ( rtIntegerArray, <resource ID> )))
```

The ExcEgg klown controls the jumpgates, and damage on a gate can affect the level's boss. Before a boss arrives on a level, its health decreases though a timer mechanism before its arrival. Thus, the longer the player delays the arrival of the boss by damaging the gate, the lower the boss's health can drop.

The computation of the health is based on interpolating time keys with percentage of health. The boss has a set of time keys, with the boss's health percentage specified for each time key. For any specific time, the boss's current health percentage is interpolated (and displayed on the boss health meter).

The gate AI in egg.r points to the resource that specifies the time keys and health percentages:

- **`BossHealthChart`** (resource; no default) specifies the `rtIntegerArray` resource that contains the time keys and health percentages.

Example:

```
AIDataValue ( "BossHealthChart", Resource, ( Val = ( rtIntegerArray, GATE_29 )))
```

See Boss Health & Gate Damage in Common Boss Behaviors for more details.

11 Crystals, Asteroids, & Moons

11.1 Crystal Klown

- *Klown Code File:* crystal.fsm
- *Klown:* Crystal
  —*Type:* CHFSM
  —*Uses External Klowns:* MoveEngines, OrientEngines, SetTarget, Scorable, ElectronConductive
  —*Has Internal Klown:* CrystalBehavior

AIDataValue ("shellDeathSequence", Resource, (Val = (rtSequence, *<resource ID>*)))

Crystals float in space and can be gathered by the player ship or grabbed by workers and taken to the gate. When taken to the gate, a crystals morphs its form and remains positioned at the gate until its is destroyed or until the gate is triggered. Crystals are controlled by:

- **value** (integer; default is 1) is the number of crystal energy units the crystal contains. For example, if the player ship grabs a crystal with three units of energy, then three units of energy is added to the ship's crystal inventory.
- **deathSequence** (resource; no default) is the sequence that plays when a crystal (other than one attached to the gate, see `shellDeathSequence` below) is killed (in *Sinistar*, this occurs when the player ship eats the crystal).
- **Model Change** (resource; no default) is the sequence that plays when a crystal delivered to the jumpgate morphs into its gate form.
- **shellDeathSequence** (resource; no default) is the sequence that plays when a morphed crystal attached to the gate explodes when the gate is shot with a Sinibomb. To change the explosion sequence, change **shellDeathSequence** for each type of crystal (in yummy.r) used in the game.

Crystals are hardcoded to ignore collisions with Sinibombs, shields, and shots.

Example:

AIDataValue ("shellDeathSequence", Resource, (Val = (rtSequence, SEQ_BOSS_TURRET_EXP)))

11.2 Asteroids (Roid Klown)

- *Klown Code File:* roid.fsm
- *Klown:* Roid
  —*Type:* CHFSM
  —*Uses External Klowns:* Health, Collision, ElectronConductive, Scorable, Difficulty, DeathPowerup
  —*Has Internal Klown:* EmitCrystals

Asteroids float about in space and can emit crystals when damaged or destroyed. Most crystal emission depends upon parameters (see below), which can be set to prevent asteroids from emitting crystals at all. (In practice, power-up roids are set to emit no crystals, and most other roids are set to emit varying degrees of crystals.) Roids that can emit crystals are hardcoded to emit at least one crystal when they are destroyed, even if under normal conditions they wouldn't.

The EmitCrystals klown also responds to messages asking about the estimated number of crystals an asteroid has. This functionality is used, for example, by Grinder when it looks for the richest roids to go mine.

11.2.1 Crystal Emission Using Difficulty Levels

```
AIDataValue ( "minDamagePerCrystal", Float,       ( Val = <#> ))
AIDataValue ( "maxDamagePerCrystal", Float,       ( Val = <#> ))
```

The amount of damage (on average) that it takes to cause a roid to emit a crystals is tied to difficulty settings:

- **minDamagePerCrystal** (floating point; default is -1.0 [means not used]) is the amount of damage (on average) a roid must take before it emits a crystal on the easiest difficulty setting.
- **maxDamagePerCrystal** (floating point; default is -1.0 [means not used]) is the amount of damage (on average) a roid must take before it emits a crystal on the hardest difficulty setting.

For other difficulty levels, the actual number used for crystal emission is scaled linearly between the minimum and maximum settings.

Note that for `minDamagePerCrystal` and `minDamagePerCrystal`, smaller values for a setting mean **more** crystals are emitted. For example, a setting of 15 means that one crystal is emitted on average per 15 points of damage done to a roid, while 30 means one crystals per 30 points of damage.

Example:

```
AIDataValue ( "minDamagePerCrystal", Float,       ( Val = 15.0 ))
AIDataValue ( "maxDamagePerCrystal", Float,       ( Val = 60.0 ))
```

11.2.2 Crystal Emission Ignoring Difficulty Levels

```
AIDataValue ( "damagePerCrystal",    Float,       ( Val = <#> ))
```

If **minDamagePerCrystal** and **maxDamagePerCrystal** (see above) are not specified, then crystal emission depends upon a parameter that ignores difficulty level:

The amount of damage (on average) that it takes to cause a roid to emit a crystals is tied to difficulty settings:

- **damagePerCrystal** (floating point; default is 0.0) is the amount of damage (on average) a roid must take before it emits a crystal, on any difficulty setting. *Note:* The code is set up so that the default value causes no crystals to be emitted.

Note that for `damagePerCrystal`, smaller values (that are above 0.0) for a setting mean **more** crystals are emitted. For example, a setting of 15 means that one crystal is emitted on average per 15 points of damage done to a roid, while 30 means one crystals per 30 points of damage.

Example:

```
AIDataValue ( "damagePerCrystal",    Float,       ( Val = 35.0 ))
```

11.2.3 Crystal Emission Sequence

```
AIDataValue ( "crystalSequence", Resource, ( Val = ( rtSequence, <resource ID> )))
```

A sequence plays when an asteroid emits crystals. The following paramenter in the roid's AI points to this sequence:

- **`crystalSequence`** (resource; no default).

Example:

```
AIDataValue ( "crystalSequence", Resource, ( Val = ( rtSequence, DEFAULT_ROID_CRYSTALS )))
```

11.3 Mining Moons (MineAsteroid Klown)

- *Klown Code File:* minemoon.fsm
- *Klown:* MineAsteroid
  —*Type:* CHFSM
  —*Uses External Klowns:* Health, TimeDamageLimiter, Collision, ElectronConductive, Scorable, Team, TargetOffset, BossOnRadar
  —*Has Internal Klown:* Main

MineAsteroids is the mining colony klown. It works mostly like a 'roid except for the way it deals with health: Through its external TImeDamageLimiter klown, it will not lose more than a certain percentage of health per some given amount of time. Also, its sensing type is assigned as `kWorker` to have it show up in team-based color on the Long-Range Radars as a non-threatening object. Its Main klown has no tunable parameters.

12 The Player Ship

12.1 Player Klown

- *Klown Code File:* player.fsm
- *Klown:* Player
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, Collision, PlayerCamera, PlayerInput, PlayerInputEngines, PlayerTarget, Weapons, Team, ShipModel, ElectronConductive, Inventory, StartWithLRRadar, SetTarget, XHair, ForceField, Scorable, Scorer, EggTractorable, AutoHealth, ShipUpgrade, CrystalMagnet, PlayerWeaponSequence
  —*Has Internal Klown:* PlayerGuts

The Player klown controls the operation of the player ship, through its internal PlayerGuts klown and through the numerous external klowns it also calls upon.

The PlayerGuts klown handles many aspects of the player ship, including:

- Managing messages from and to mission control concerning the player ship.
- Managing the health veins.
- Managing the weapons inventory UI, such as when the player switches weapons, changing weapons' ammo displays as necessary, removing temporary weapons when they're used up, adding new weapons.
- Loading in the correct bucket (special items inventory) for the ship.
- Managing the number of lives display UI, when lives are added or lost.
- Managing the crystal UI when crystals are added or expended.
- Turning on autoaiming and the crystal magnet if the player is playing Easy difficulty.
- Managing the ship's state when the ship is upgraded or killed, so that the next ship will inherit the correct state.
- Running the death throes when the ship dies.
- Being aware of an activated cloak and the invulnerability cheat.
- Being aware if force feedback is on.

12.1.1 Lives

```
AIDataValue ( "easyNumLives",     Integer,   ( Val = <#> ))
AIDataValue ( "mediumNumLives",   Integer,   ( Val = <#> ))
AIDataValue ( "hardNumLives",     Integer,   ( Val = <#> ))
AIDataValue ( "numLives",         Integer,   ( Val = <#> ))
```

Four parameters, one for each difficulty level, allow you to specify the number of lives with which the player starts the game:

- **easyNumLives** (integer; default is 1) specifies the number of lives the player starts with on the Easy difficulty level.
- **mediumNumLives** (integer; default is 1) specifies the number of lives the player starts with on the Normal difficulty level.
- **hardNumLives** (integer; default is 1) specifies the number of lives the player starts with on the Hard difficulty level.
- **numLives** (integer; default is 1) specifies the number of lives the player starts with on the Sinistar difficulty level.

Example:

```
AIDataValue ( "easyNumLives",        Integer,  ( Val = 9 ))   // For Easy level
AIDataValue ( "mediumNumLives",      Integer,  ( Val = 5 ))   // For Normal level
AIDataValue ( "hardNumLives",        Integer,  ( Val = 3 ))   // For Hard level
AIDataValue ( "numLives",            Integer,  ( Val = 3 ))   // For Sinistar level
```

12.1.2 Health Veins

AIDataValue ("healthVeins", Resource, (Val = (rtHealthVeins, *<resource ID>*)))

The player ship has a set of veins, which visually pulse to show the player the ship's current health—the color of the pulses indicates how healthy or injured the ship is. These pulses are controlled by an `rtHealthVeins` resource. In the player ship's AI, point to this resource using:

- **healthVeins** (resource; no default) specifies the `rtHealthVeins` resource the ship uses to control its health veins.

Example:

```
AIDataValue ( "healthVeins",   Resource,   ( Val = ( rtHealthVeins, PLAYER )))

defres rtHealthVeins (PLAYER)
(
   Name = "*/vains inside"
   Health[] =
   (
      health    = 0.1
      color1    = RGBCOLOR( 190, 5, 1  )
      color2    = RGBCOLOR(  0,  0, 0  )
      pulseTime =  0.3
      offset    = -0.5
   )
   Health[] =
   (
      health    = 0.2
      color1    = RGBCOLOR( 252, 66, 1  )
      color2    = RGBCOLOR( 0, 0, 0  )
      pulseTime = 0.55
      offset    =  -0.5
   )
   ...
   )
)
```

12.1.3 Low Health Sequence

```
AIDataValue ( "lowHealthSeq",   Resource,   ( Val = ( rtSequence, <resource ID> ))) 
```

A sequence is triggered when the player ship's health drops below 33% of its maximum possible health. In the player ship's AI, point to this sequence using:

- **lowHealthSeq** (resource; no default) specifies the `rtSequence` resource that plays when the ship's health drops to or below its low health level.

Note: The code is structured so that the sequence won't be triggered repeatedly if the ship's health fluctuates across its low health range (such as the ship repeatedly healing just above the 33% mark but immediately taking damage again). Instead, once the sequence is triggered, it cannot play again until after the ship's health equals or exceeds 60% of its maximum possible health.

Example:

```
AIDataValue ( "lowHealthSeq",   Resource,   ( Val = ( reSequence, TIMMY1_LOWHEALTH ))) 
```

12.1.4 Buckets (Inventory Slots)

```
AIDataValue ( "BucketName",         String,   ( Val = "<name>" ))
```

Each player ship can have a different number of slots for the specials inventory. To set the slots, create an inventory bucket in yummy.r:

```
defres rtInvBucket (BUCKET_1)
(
   Name        = "bucket 1"
   MaxCrystals = 100
   NumSlots    = 4
)
```

NumSlots is the number of inventory slots.

In the `rtAI` resource of each player ship, point the following parameter to the inventory bucket to use:

- **BucketName** (string; no default) specifies the name of the bucket the ship uses.

Example:

```
AIDataValue ( "BucketName",         String,   ( Val = "bucket 1" ))
```

12.1.5 Auto-Aiming (for Easy Difficulty)

```
AIDataValue ( "easyAutoAimAngle",        Float,    ( Val = <#> ))
```

When playing on the Easy difficulty level (only) the player ship auto-aiming, which can affect (only) the mining laser, the concussion shot, and the mind control shot (as of this writing, only the mining laser is affected). When the player fires an auto-aiming weapon on a target that is within the auto-aiming cone (which extends from the front of the player ship), the shot auto-aims using leading to try to hit the target. The following parameter defines the auto-aiming cone:

- **easyAutoAimAngle** (floating point; default is 180.0) specifies the angle (in degrees) of the auto-aiming cone.

Example:

```
AIDataValue ( "easyAutoAimAngle",        Float,    ( Val = 10.0 ))
```

For the auto-aiming system to do its leading computations, it must know the speed of the shot, so add a `shotSpeed` parameter to the `rtAI` resource of each weapon (in playerweapon.r) that uses auto-aiming:

- **shotSpeed** (floating point; default is 0.0) tells the auto-aiming system the speed of the shot. (The default value means auto-aiming is not used for the weapon.)

Note that `shotSpeed` here does ***not*** set the speed of the shot—it only tells the auto-aiming system what the speed is. The shot's speed is set (as before) in the emitter for the firing sequence, so make sure you check this value in the emitter and reuse it for `shotSpeed`.

Example:

```
defres rtAI ( PW_LASER1 )
(
   Type.Klown = (KlownName = "TimedWeapon")
   AIDataValue ( "Category",     Integer,  ( Val  = GUNCAT_LASER ))
   AIDataValue ( "GunportNames", Resource, ( Val  = ( rtGunportNames, PW_LASER1 )))
   AIDataValue ( "FireSeq",      Resource, ( Val  = ( rtSequence, PW_LASER1_SEQ1 )))
   AIDataValue ( "AveSeq",       Resource, ( Val  = ( rtSequence, PW_LASER1_SEQ2 )))
   AIDataValue ( "Delay",        Float,    ( Val  = 0.12 ))
   AIDataValue ( "shotSpeed",    Float,    ( Val  = 1200 ))
)
```

12.2 ShipUpgrade Klown

- *Klown Code File:* shipupgrade.fsm
- *Klown:* ShipUpgrade
  —*Type:* Submachine

```
AIDataValue ( "shipNum",           Integer,    ( Val = <#> ))
AIDataValue ( "UpgradeSequence",   Resource,   ( Val = ( rtSequence, <resource ID> ))
```

Place the following parameters in the player ship AI resources (in timmy.r) to specify upgrade information:

- **shipNum** (integer; default is 1) specifies the level of the ship to use (1 is Timmy 1, 2 is Timmy 2, etc.).
- **UpgradeSequence** (resource; no default) specifies the `rtSequence` resource that plays on the player ship when it is upgraded.

Example:

```
AIDataValue ( "shipNum",           Integer,    ( Val = 3 ))
AIDataValue ( "UpgradeSequence",   Resource,   ( Val = ( rtSequence, SEQ_TIMMY3_UPGRADE ))
```

12.3 Turbocharged Crystal Magnet (for Easy Difficulty)

- *Code File:* zoom.cpp
- *Klown Code File:* none
- *Klown:* none

```
AIDataValue ( "crystalSensor",      Resource, ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "springMaxToAffect",  Integer,  ( Val = <#> ))
AIDataValue ( "springMaxPullDist",  Float,    ( Val = <#> ))
AIDataValue ( "springReleaseSpeed", Float,    ( Val = <#> ))
AIDataValue ( "springConstant",     Float,    ( Val = <#> ))
AIDataValue ( "springMaxForce",     Float,    ( Val = <#> ))
AIDataValue ( "springDragTime",     Float,    ( Val = <#> ))
```

The player ship has a much more aggressive crystal magnet that is active in only in the Easy difficulty level. Pretty much, it just snacks on crystals and you can't escape them. The AI data values to control this are located in `rtAI` for each player ship. For tuning purposes, three are important:

- **`crystalSensor`** (resource; default is no sensor) specifies the type of detector the crystal magnet uses to detect crystals. See Sensors for more details on sensors.
- **`springMaxToAffect`** (integer; default is 5) specifies the maximum number of crystals that can simultaneously be dragged in.
- **`springMaxPullDist`** (floating point; default is 400.0) specifies the maximum distance (in meters) at which the crystals will be pulled in.

Four other parameters control how the `cSpringBehavior` class in zoom.cpp moves the affected crystals; you probably won't need to tune these:

- **`springReleaseSpeed`** (floating point; default is 20.0).
- **`springConstant`** (floating point; default is 2.5).
- **`springMaxForce`** (floating point; default is 500.0).
- **`springDragTime`** (floating point; default is 1.0).

Example:

```
AIDataValue ( "crystalSensor",      Resource, ( Val = ( rtBodySensor, TIMMY_CRYSTALSENSOR
                ) ))
AIDataValue ( "springMaxToAffect",  Integer,  ( Val =   2 ))
AIDataValue ( "springReleaseSpeed", Float,    ( Val =  50 ))
AIDataValue ( "springMaxPullDist",  Float,    ( Val = 600 ))
AIDataValue ( "springConstant",     Float,    ( Val =  50 ))
AIDataValue ( "springMaxForce",     Float,    ( Val = 500 ))
AIDataValue ( "springDragTime",     Float,    ( Val =   1 ))
```

12.4 Specials

12.4.1 CrystalDepot Klown

- *Klown Code File:* crystaldepot.fsm
- *Klown:* CrystalDepot
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Team, Health, Match Engines, Collision

```
AIDataValue ( "crystalSensor",   Resource,   ( Val = ( rtBodySensor, <resource ID> )))
```

The player can drop off a Crystal Depot special in space, and the depot will attract nearby crystals and place them in orbit around the depot. The depot is hardcoded: 1) to be invulnerable to damage and 2) to have "brakes" that always work to stop any lateral motion it may acquire. It uses the following parameter:

- **`crystalSensor`** (resource; no default) specifies the sensor the depot uses to detect crystals. See Sensors for more details on sensors.

Example:

```
AIDataValue ( "crystalSensor",   Resource,   ( Val = ( rtBodySensor, CRYSTAL_DEPOT )))
```

12.4.2 Decoy Klown

- *Klown Code File:* decoy.fsm
- *Klown:* Decoy
  —*Type:* CHFSM
  —*Uses External Klowns:* Birth, Health, ScoreVector
  —*Has Internal Klown:* DecoyGuts

One possible special in the game was a decoy bomb, but it did not make the cut for the game. The Decoy klown was set up to handle the functioning of decoys. It has no tunable AI parameters.

12.5 Weapon Selection Seqences

- *Klown Code File:* player.fsm
- *Klown:* PlayerWeaponSequence

```
AIDataValue ( "weaponSelSeqs",   Resource,   ( Val = ( rtIntegerArray, <resource ID> )))
```

The PlayerWeaponSequence klown is a submachine that handles which sequence is played when the player selects a weapon on the weapons inventory:

- **`weaponSelSeqs`** (resource; no default) points to an `rtIntegerArray` resource that lists the weapon-selection sequence for each weapon on the weapons inventory. A weapon's sequence in the `rtIntegerArray` resource must appear in the same order as the weapon is listed in the weapon categories.

Example:

```
AIDataValue ( "weaponSelSeqs", Resource, ( Val = ( rtIntegerArray, SEQ_TIMMY_ALL_WEAPONS
                ) ))
```

```
defres rtIntegerArray( SEQ_TIMMY_ALL_WEAPONS )
(
   Int[] = ( Value = SEQ_TIMMY_WEAPON1 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON2 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON3 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON4 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON5 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON6 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON7 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON8 )
   Int[] = ( Value = SEQ_TIMMY_WEAPON9 )
)
```

13 Meters and Displays

13.1 Long-Range Radar (LRRadar Klown)

- *Klown Code File:* LRRadar.fsm
- *Klown:* LRRadar
  —*Type:* HFSM

```
AIDataValue ( "LRRadarSensor",       Resource, ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "LRRadarNoRoidSensor", Resource, ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "DisplayRange",        Float,    ( Val = <#> ))
AIDataValue ( "CrystalColor",        Integer,  ( Val = <RGB macro> ))
AIDataValue ( "WarriorColorMod",     Float,    ( Val = <#> ))
AIDataValue ( "BossColorMod",        Float,    ( Val = <#> ))
```

See the *Sinistar* Play Manual for an overview of the Long-Range Radar. These parameters control the radar:

- **LRRadarSensor** (resource; default is no sensor) is the sensor the radar uses to detect all objects, including asteroids and crystals. See Sensors for more details on sensors.
- **LRRadarNoRoidSensor** (resource; default is no sensor) is the sensor the radar uses to detect objects except asteroids and crystals. See Sensors for more details on sensors.
- **DisplayRange** (floating point; default is 5000.0) is the maximum distance (in meters) from the player ship at which the radar detects objects.
- **CrystalColor** (integer using RGBCOLOR macro; default is RGB(192, 192, 0)) is the RGB color used to display asteroids and crystals on the radar.
- **WarriorColorMod** (floating point; =>0.0, =<1.0; default is 0.3) allows the display color of detected warriors to be modified from their basic team color. The object's team color is mutliplied by this modifier, thereby shifting the color.
- **BossColorMod** (floating point; =>0.0, =<1.0; default is 0.5) allows the display color of detected bosses to be modified from their basic team color. The object's team color is mutliplied by this modifier, thereby shifting the color.

Example:

```
AIDataValue ( "LRRadarSensor",       Resource, ( Val = ( rtBodySensor, TIMMY_LRRADAR ))
AIDataValue ( "LRRadarNoRoidSensor", Resource, ( Val = ( rtBodySensor,
                TIMMY_LRRADAR_NOROID )))
AIDataValue ( "DisplayRange",        Float,    ( Val = 5000.0 ) )
AIDataValue ( "CrystalColor",        Integer,  ( Val = RGBCOLOR(175, 212, 227) ))
AIDataValue ( "WarriorColorMod",     Float,    ( Val = 0.4 ))
AIDataValue ( "BossColorMod",        Float,    ( Val = 0.7 ))
```

13.2 Short-Range Radar (SRRadar Klown)

- *Klown Code File:* srradar.fsm
- *Klown:* SRRadar
  —*Type:* HFSM

```
AIDataValue ( "SRRadarSensor",       Resource, ( Val = ( rtBodySensor, <resource ID> )))
AIDataValue ( "MinTraceTime",        Float,    ( Val = <#> ))
AIDataValue ( "DeathFadeTime",       Float,    ( Val = <#> ))
AIDataValue ( "MaxTraces",           Integer,  ( Val = <#> ))
AIDataValue ( "MaxDyingTraces",      Integer,  ( Val = <#> ))
AIDataValue ( "TargetColor",         Integer,  ( Val = <RGB macro> ))
AIDataValue ( "DeathStartColor",     Integer,  ( Val = <RGB macro> ))
AIDataValue ( "DeathEndColor",       Integer,  ( Val = <RGB macro> ))
AIDataValue ( "BossColorMod",        Float,    ( Val = <#> ))
AIDataValue ( "AttackerColorMod",    Float,    ( Val = <#> ))
AIDataValue ( "ReticuleTextureFile", String,   ( Val = "<string>" ))
AIDataValue ( "StdReticule",         String,   ( Val = "<string>" ))
AIDataValue ( "TargetReticule",      String,   ( Val = "<string>" ))
AIDataValue ( "BossReticule",        String,   ( Val = "<string>" ))
```

See the *Sinistar* Play Manual for an overview of the Short-Range Radar (the Targeting Radar). In short, the Short-Range Radar displays reticules on the screen for:

- The object the radar's host object is currently targeting. (The host object is whatever object the radar is attached to; in *Sinistar* this is always the player ship.)
- All bosses currently in the game.
- The jumpgate.
- Other objects it can sense that that are in detection range, up the limits of its available radar traces.

The reticule for an objects uses the object's team color, although this can be modified or overridden by the radar's parameters. When an objects being tracked by the radar is destroyed, its reticule can fade between two colors over a period of time before the reticule is removed.

These parameters control the radar:

- **`SRRadarSensor`** (resource; default is no sensor) is the sensor the radar uses to detect all objects, including asteroids and crystals. See Sensors for more details on sensors.
- **`MinTraceTime`** (floating point; default is 2.0) is the minimum time (in seconds) a reticule is displayed for a tracked object. (This helps prevent the radar from changing reticules too quickly when there are many objects in range.)
- **`DeathFadeTime`** (floating point; default is 3.0) is the time (in seconds) a reticule is displayed for a destroyed object.
- **`MaxTraces`** (integer; default is 6) is the maximum number of objects for which the radar will simultaneously display reticules. (For example, `Val = 6` means the radar will display a maximum of six reticules.) If there are more than `MaxTraces` objects being tracked, the objects selected for the available `MaxTraces` reticules are chosen per the following priorities:
  1. The object is the host object's current target.
  2. The object is a boss.
  3. The object has only been in the radar system for a short while.

4. The objects nearest to the radar's host object.

- **MaxDyingTraces** (integer; default is 8) is the maximum number of destroyed objects for which the radar will simultaneously display fading "death" reticules.
- **TargetColor** (integer using RGBCOLOR macro; default is RGB(255, 255, 255)) is the RGB color used to display the object the host object is currently targeting.
- **DeathStartColor** (integer using RGBCOLOR macro; default is RGB(255, 0, 0)) and **DeathEndColor** (integer using RGBCOLOR macro; default is RGB(32, 32, 32)) are the initial and final RGB colors used to display an object that has been destroyed. The reticule's color fades smoothly between these colors during the `DeathFadeTime`.
- **BossColorMod** (floating point; =>0.0, =<1.0; default is 0.5) allows the display color of bosses to be modified from their basic team color. The object's team color is multiplied by this modifier, thereby shifting the color.
- **AttackerColorMod** (floating point; =>0.0, =<1.0; default is 0.5) allows the display color of detected attackers to be modified from their basic team color. The object's team color is multiplied by this modifier, thereby shifting the color.
- **ReticuleTextureFile** (string; no default) is the name of the file that contains the textures for the reticules.
- **StdReticule** (string; no default) is the name of the texture for the standard reticule.
- **TargetReticule** (string; no default) is the name of the texture for the current-target reticule.
- **BossReticule** (string; no default) is the name of the texture for the boss reticule.

Note: The SRRadar klown also contains the code for the "cheats" that cycle through and take snapshots of the AI states of objects. See the *Sinistar: Unleashed* Release Notes for details on this cheat.

Example:

```
AIDataValue ( "SRRadarSensor",      Resource,  ( Val = ( rtBodySensor, TIMMY1_RADAR )))
AIDataValue ( "MinTraceTime",       Float,     ( Val = 0.5 ))
AIDataValue ( "DeathFadeTime",      Float,     ( Val = 1.3 ))
AIDataValue ( "MaxTraces",          Integer,   ( Val = 4 ))
AIDataValue ( "TargetColor",        Integer,   ( Val = RGBCOLOR(255, 255, 255) ))
AIDataValue ( "DeathStartColor",    Integer,   ( Val = RGBCOLOR(255,  64,  64) ))
AIDataValue ( "DeathEndColor",      Integer,   ( Val = RGBCOLOR( 32,  32,  32) ))
AIDataValue ( "BossColorMod",       Float,     ( Val = 0.6 ))
AIDataValue ( "AttackerColorMod",   Float,     ( Val = 0.4 ))
AIDataValue ( "ReticleTextureFile", String,    ( Val = "ui_onscreen.tgx" ))
AIDataValue ( "TargetReticle",      String,    ( Val = "SRRTarget" ))
AIDataValue ( "BossReticle",        String,    ( Val = "SRRBoss" ))
```

14 Powerups

Note: The cost of a free life can be controlled by modifying the `ammoCost` parameter in the `rtAI` for `PW_EXTRALIFE` (located in playerweapon.r):

```
defres rtAI (PW_EXTRALIFE)
(
   Type.Klown = (KlownName = "ExtraLifeWeapon")
   AIDataValue ( "ammoCost",  Integer,  ( Val  = 60 ))
   AIDataValue ( "Category",  Integer,  ( Val  = GUNCAT_EXTRALIFE ))
)
```

]]

15 Mission Control

15.1 MissionGoals Klown

- *Klown Code File:* missiongoals.fsm
- *Klown:* MissionGoals
  —*Type:* HFSM

```
AIDataValue ( "MissionList",   Resource,   ( Val = ( rtMissionInfo, <resource ID> ))) 
AIDataValue ( "MissionCookie", Resource,   ( Val = ( rtSequence, <resource ID> ))) 
AIDataValue ( "surviveCount",  Integer,    ( Val = <#> )) 
AIDataValue ( "affectBonus",   Integer,    ( Val = <#> )) 
AIDataValue ( "SeqPlayPoint",  Integer,    ( Val = <resource ID> ))
```

The MissionGoals klown allows you to set mission goals for a level (it's used for bonus levels in *Sinistar*). It has the following tunable parameters:

- **MissionList** (resource; no default) points to a `rtMissionInfo` resource that specifies a list of objects in the level and, for each object, whether it should live or die for the mission to succeed. See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm for further details on `rtMissionInfo`.
- **MissionCookie** (resource; no default) points to a `rtSequence` resource that plays when the missions goals are fulfilled.
- **surviveCount** (integer; default is 0), for a mission in which one or more objects should live, sets how many objects must survive for the mission to succeed. If fewer objects survive, the mission fails.
- **affectBonus** (integer; default is 0), for a successful mission in which several objects should live, sets whether the mission's bonus is scored if one or more objects lives or is scored separately for each object that lives. For example, say there are four mining colonies that should live. If `Val = 0`, then the mission bonus is scored if one or more of the colonies survive. If `Val = 1`, then the mission bonus is scored for each colony that survives (in this example, the bonus could be scored four times if all four colonies survive).
- **SeqPlayPoint** (integer (a resource ID); no default) allows a sequence to be played centered on a container. Both the sequence and the container must use the same resource ID. For example, `Val = MINEMOON1XA` for `SeqPlayPoint` causes an `rtSequence ( MINEMOON1XA )` sequence to play on a `rtContainer ( MINEMOON1XA )`. *Note:* This parameter was not used in *Sinistar*.

Example:

```
AIDataValue ( "MissionList",   Resource, ( Val = ( rtMissionInfo, MISSION_DESIGN_GROUP1
                ))) 
AIDataValue ( "MissionCookie", Resource, ( Val = ( rtSequence,    MISSION_DESIGN_GROUP2
                ))) 
AIDataValue ( "surviveCount",  Integer,  ( Val = 1)) 
AIDataValue ( "affectBonus",   Integer,  ( Val = 1))
```

15.2 Mission Status

15.2.1 MissionStatus Klown

- *Klown Code File:* missionstatus.fsm
- *Klown:* MissionStatus
  —*Type:* Submachine

```
AIDataValue ( "levelSwitchDelay",       Float,   ( Val = <#> ))
AIDataValue ( "missionFailedDelay",     Float,   ( Val = <#> ))
AIDataValue ( "gateDestroyedDelay",     Float,   ( Val = <#> ))
AIDataValue ( "generatorDestroyDelay", Float,   ( Val = <#> ))
AIDataValue ( "powerupConsumedDelay",  Float,   ( Val = <#> ))
AIDataValue ( "nextLevel",              Float,   ( Val = <#> ))
AIDataValue ( "BonusLevel",             Float,   ( Val = <#> ))
AIDataValue ( "NumGoals",               Float,   ( Val = <#> ))
```

Various delays can occur between when the player succeeds or fails in meeting an objective and when the next event (such as the score screen at the end of the level) occurs:

- For a failed mission, **`missionFailedDelay`** (floating point; default is 5.0) specifies the delay time (in seconds) until the next event occurs.
- For a successful mission, **`levelSwitchDelay`** (floating point; default is 25.0) specifies the delay time (in seconds) until the next event occurs.
- When the player destroys a jumpgate, **`gateDestroyedDelay`** (floating point; default is 10.0) specifies the delay time (in seconds) until the next event occurs.
- When the player destroys a shield generator on a station, **`generatorDestroyDelay`** (floating point; default is 30.0) specifies the delay time (in seconds) until the next event occurs.
- When the player consumes a special power-up, **`powerupConsumedDelay`** (floating point; default is 4.0) specifies the delay time (in seconds) until the next event occurs.
- **`nextLevel`** (string; no default) specifies the next level that will run when the player is done with the current level.
- **`BonusLevel`** (integer; default is 0) specifies whether (`Val=1`) or not (any other setting) the level is a bonus level. If the level is a bonus level, the player does not restart it if he/she fails his/her mission.
- **`NumGoals`** (integer; default is 0) specifies the number of goals the player has to accomplish to win the level.

Example:

```
AIDataValue ( "levelSwitchDelay",   Float,     ( Val = 60.0 ))
AIDataValue ( "missionFailedDelay", Float,     ( Val =  5.0 ))
```

NOTE

These two parameters had different functions before the score screen was added:

- **`levelSwitchDelay`** controls the amount of time (in seconds) between when the mission succeeds and the level advances. On levels where the player gets sucked into the gate, the level advances immediately once the player arrives at the center of the gate and it advances when `levelSwitchDelay` times out if the player can manage to avoid being sucked in by the tractor beam (this should be impossible to do... but you never

know). So on gate levels it should be a fairly long delay. On non-gate levels (8, 12 etc) the delay should be like it was at around 10 seconds.

- **missionFailedDelay** is likewise the delay between when the player gets killed or the mission otherwise fails and the level reloads.

15.3 Player Setup

```
defres rtAI ( MISSION_SETUP )
(
...
   // Where player starts in level
   AIDataValue ( "playerPosition", Vector, ( Val = ( <#>, <#>, <#> )))
...
)
```

The mission setup AI contains a parameter to position the player:

- **playerPosition** specifies the position (in world coordinates) at which the player begins at the start of the level.

Example:

```
defres rtAI ( MISSION_SETUP )
(
...
   // Where player starts in level
   AIDataValue ( "playerPosition", Vector, ( Val = ( 0, -2500, 0 )))
...
)
```

15.4 Warrior Setup

```
defres rtSequence ( MISSION_SETUP )
(
...
   // What warriors are emitted at start of level (see below)
   Actions[] = ( ActionId = ( rtEmitInfo,  <resource ID> ) Time = 0 )
...
)
```

The mission setup starting sequence (which is run by the mission setup AI) contains an action that specifies the emitter for the initial warriors (and/or other ships or containers) for the level:

- ***<resource ID>*** specifies the `rtEmitInfo` resource that emits the initial warriors at the start of the level.

Example:

```
defres rtSequence ( MISSION_SETUP )
(
...
   // What warriors are emitted at start of level (see below)
   Actions[] = ( ActionId = ( rtEmitInfo,  WARRIORS1   ) Time = 0 )
...
```

```
)
```

The `rtEmitInfo` resource is a standard `rtEmitInfo` resource (see the separate document, Emitters).

Example:

```
defres rtEmitInfo ( WARRIORS1 )
(
   NumParticles       = 6                                  // Number of particles
   EmitTime           = 0.0                                // Time to emit particles
   Emitter.EmitSingle = ( ContainerId = TICK3 )            // Type of tick to emit
   InitInfo[] =
   (
      Info.InitPosition =                                  // Position of emission
      (
         EmitSpace          = 0                           // 0 = object space
         Type.EmitPointList =                             // Emit at list of points
         (
            Usage   = 1                                   // Emit sequentially
            Point[] = (Val = (   500,  500, 0 ))
            Point[] = (Val = (   500, -500, 0 ))
            Point[] = (Val = (  -500,  500, 0 ))
            Point[] = (Val = (  -500, -500, 0 ))
            Point[] = (Val = (  1000,    0, 0 ))
            Point[] = (Val = ( -1000,    0, 0 ))
         )
      )
   )
   INIT_ORIENT_CENTERFACING()                             // Orientation of emission
)
```

15.5 Powerup Transports

```
AIDataValue ( "transportId",            Integer,  ( Val = <resource ID> ))
AIDataValue ( "transportDelay",         Float,    ( Val = <#> ))
AIDataValue ( "transportInitialDelay",  Float,    ( Val = <#> ))
AIDataValue ( "transportSpawnDist",     Float,    ( Val = <#> ))
AIDataValue ( "transportInnerDist",     Float,    ( Val = <#> ))
```

The non-bonus levels can have powerup-carrying transports emitted during play of the level. To control this, use the following parameters in the `MISSION_SETUP` for each level:

- **`transportId`** (resource ID) specifies the resource ID of the transport's container.
- **`transportDelay`** (floating point) specifies the time (in seconds) to wait between emitting a transport.
- **`transportInitialDelay`** (floating point) specifies the time (in seconds) to wait from the start of the level before emitting the first transport.
- **`transportSpawnDist`** (floating point) specifies the distance (in meters) from the gate where the transport is emitted. It is emitted at a random point on the sphere with a radius of `transportSpawnDist`.
- **`transportInnerDist`** (floating point) specifies the distance (in meters) that the transport must approach the gate before heading out into deep space.

Note: Transport creation is hooked into all levels except the bonus levels.

Example:

```
AIDataValue ( "transportId",            Integer,  ( Val = LITETRANS ))
AIDataValue ( "transportDelay",         Float,    ( Val =   90.0 ))
AIDataValue ( "transportInitialDelay", Float,     ( Val =   30.0 ))
AIDataValue ( "transportSpawnDist",     Float,    ( Val = 5000.0 ))
AIDataValue ( "transportInnerDist",     Float,    ( Val = 1500.0 ))
```

15.6 The Powerups of Death

```
AIDataValue ( "emitDeathPowerup",  Integer,  ( Val = <#> ))
```

In theory, any object in the game can emit a powerup when it is destroyed. Currently, the code supports transports and asteroids emitting powerups. To have an object emit a powerup when the object is destroyed, add the following parameter to the object's `rtAI` resource:

- **`emitDeathPowerup`** (integer; 0 or 1; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) a powerup is emitted when the object dies.

Example:

```
AIDataValue ( "emitDeathPowerup",  Integer,  ( Val = 1 ))
```

15.7 Powerups in Space

```
AIDataValue ( "powerupId", Integer, ( Val = <resource ID> ))
```

To create a powerup container that sits in space, make a container with an `rtAI` property using the `JBBestow` klown type. All the standard jellybaby parameters can be used (see the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm). The one additional parameter is:

- **`powerupId`** (resource ID; no default) specifies the ID of a jellybaby (powerup) that gets added to the player's inventory once the player collides with the powerup container.

Example:

```
defres rtAI ( POWERUP_GENERIC )
(
   Type.Klown = ( KlownName = "JBBestow" )
   AIDataValue ( "powerupId", Integer, ( Val = JB_ORBITAL1 ))
   ...
)
```

15.8 Powerup Emission

```
AIDataValue ( "mustEmitPowerups", Resource, ( Val = ( rtMustEmitPowerups, <resource ID>
                  )))
```

```
AIDataValue ( "randomPowerups",   Resource, ( Val = ( rtRandomPowerups,   <resource ID>
                  )))
```

Each time a powerup is emitted, a selection process is performed to decide what to emit. The mission control AI data has two parameters that list mandatory and randomly emitted powerups:

- **mustEmitPowerups** (resource; no default) specifies the `rtMustEmitPowerups` resource for the level. This resource can list up to three powerups, and these powerups will be the first three powerups emitted on the level.
- **randomPowerups** (resource; no default) specifies the `rtRandomPowerups` resource for the level. This resource can list any number of powerups, and these powerups will be emitted randomly (per their assigned weights) whenever powerups are emitted after all `rtMustEmitPowerups` powerups are emitted.

Note: See the separate document, CZ Types: Resource Types Specific to *Sinistar: Unleashed*http://intranet.gamefx.com/intranet/reference/gfx_doc/manuals/cztypes/cztypes.htm, on how `rtMustEmitPowerups` and `rtRandomPowerups` resources are defined and how they work.

Example:

```
AIDataValue ( "mustEmitPowerups", Resource,  ( Val = ( rtRandomPowerups, MISSION_SETUP )))
AIDataValue ( "randomPowerups",   Resource,  ( Val = ( rtRandomPowerups, MISSION_STD_YUMMY
                  )))

defres rtMustEmitPowerups ( MISSION_STD_YUMMY )
(
   Powerup[] = ( ContainerId = POWERUP_CRYSTALDEPOT )
   Powerup[] = ( ContainerId = POWERUP_ORBITAL )
   Powerup[] = ( ContainerId = POWERUP_T_MINDCONTROL )
)

defres rtRandomPowerups ( MISSION_STD_YUMMY )
(
   Powerup[] = ( ContainerId = POWERUP_SCORE_100   Weight = 5 )
   Powerup[] = ( ContainerId = JB_SHIPUPGRADE      Weight = 5 DontEmitShip = 6  )
   Powerup[] = ( ContainerId = POWERUP_SRMISSILES  Weight = 5 DontEmitGuncat =
                  GUNCAT_SRMISSILE )
)
```

15.9 Cloaked Unknown Entities

- *Klown Code File:* missionleash.fsm
- *Klown:* MissionLeash

```
AIDataValue ( "leashType",            Integer,   ( Val = <flag> ))
AIDataValue ( "leashRadius",          Float,     ( Val = <#> ))
AIDataValue ( "leashCylinderAxis",    Vector,    ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "leashContainer",       Integer,   ( Val = <resource ID> ))
AIDataValue ( "leashEmitDistance",    Float,     ( Val = <#> ))
AIDataValue ( "leashTimePerEmission", Float,     ( Val = <#> ))
```

Cloaked ships lurk at the fringes of the asteroid field to drive the player back. In essence, the player is confined to a leash area centered on the center of the level, and the player is attacked by the cloaked ships when he/she ventures outside the leash area. The leash area can be spherical or cylindrical in shape. A spherical leash area leashes the player

to a specified distance from the center of the level. A cylindrical leash area has an axis specified for the level; the player is leashed to a specified distance from the axis of the cylinder, which runs the entire length of the level.

Place the following data values in the `rtAI (MISSION_CONTROL)` resource of each level (in each level_XX.r file) to cause cloaked enemies to get created once the player gets too far from the center of the level:

- **`leashType`** (flag; default is `kLeashSphere`) specifies whether the shape of the leash area is spherical (`Val = kLeashSphere`, the default) or cylindrical (`Val = kLeashCylinder`).
- **`leashRadius`** (floating point; no default) is the radius (in meters) of the leash area's sphere or cylinder.
- **`leashVector`** (vector; no default) is the axis (in world coordinates) of the cylinder. The parameter only has to be specified if the `leashType` is `kLeashCylinder`.
- **`leashContainer`** (resource ID; default is 1) is the resource ID of container of the warriors to emit once the player gets too far from the field. *Note:* Any container can be emitted, but the game uses cloaked warriors so they won't be detected by the player.
- **`leashEmitDistance`** (floating point; default is 1.0) is the distance (in meters) from the player at which enemies get emitted.
- **`leashTimePerEmission`** (floating point; default is 1.0) is the time (in seconds) between emissions of warriors once the player gets too far.

Example:

```
AIDataValue ( "leashContainer",       Integer,  ( Val = CLOAKED_TICK1 ))
AIDataValue ( "leashRadius",          Float,    ( Val = 3000.0 ))
AIDataValue ( "leashEmitDistance",    Float,    ( Val =  500.0 ))
AIDataValue ( "leashTimePerEmission", Float,    ( Val =    8.0 ))
```

15.10 Stop That Spawning!

- *Klown Code File:* excegg.fsm
- *Klown:* ExcEgg

```
AIDataValue ( "StopSpawning", Integer, ( Val = <#> ))
```

The following AI parameter causes the jumpgate to tell mission control to stop spawning workers and warriors after the boss arrives on the level:

- **`StopSpawning`** (integer; 0 or 1; default is 1) specifies whether (`Val = 1`) or not (`Val = 0`) the spawning of workers and warriors ceases when the boss arrives.

Example:

```
AIDataValue ( "StopSpawning", Integer, ( Val = 1 ))
```

15.11 Obsolete Parameter

The following parameter is no longer used:

- `destroyVictoryCond` used to specify the object that, when destroyed, signaled the level was finished. This function is now tied to gate destruction or the tractor beam if the gate is not destroyed.

16 Miscellaneous AIs

16.1 Cameras

16.1.1 Camera Klown

- *Klown Code File:* camera.fsm
- *Klown:* Camera
  —*Type:* CHFSM
  —*Uses External Klowns:* SetTarget
  —*Has Internal Klown:* CameraControl

```
AIDataValue ( "folOffsetPosition",  Vector, ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "folLagTime",         Float,  ( Val = <#> ))
AIDataValue ( "folSwitchTime",      Float,  ( Val = <#> ))
AIDataValue ( "fpOffsetPosition",   Vector, ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "fpLagTime",          Float,  ( Val = <#> ))
AIDataValue ( "fpSwitchTime",       Float,  ( Val = <#> ))
AIDataValue ( "longOffsetPosition", Vector, ( Val = ( <#>, <#>, <#> )))
AIDataValue ( "longLagTime",        Float,  ( Val = <#> ))
AIDataValue ( "longSwitchTime",     Float,  ( Val = <#> ))
AIDataValue ( "sniperZoomTime",     Float,  ( Val = <#> ))
AIDataValue ( "sniperFov",          Float,  ( Val = <#> ))
```

The camera klown controls the camera(s) in the game. In *Sinistar*, there is one camera that is tied to the player ship. It has four modes: a third-person view camera (the (follow cam) that follows the ship, a first-person view camera, a long-shot camera used when the ship is killed, and a zoom view camera (the sniper cam) used for a magnified view of a small volume of space in front of the ship. The klown uses the following parameters:

- **folOffsetPosition, fpOffsetPosition, longOffsetPosition** (vector; no default) is the offset from the player ship that the camera is positioned at, for the follow cam, first person cam, and long cam respectively.
- **folLagTime, fpLagTime, longLagTime** (floating point; no default) is the amount of time (in seconds) the camera "lags behind" the position of the player ship as it moves, for the follow cam, first person cam, and long cam respectively. Lag time values can be negative, which means the camera gets to "see" things before the ship does.)
- **folSwitchTime, fpSwitchTime, longSwitchTime** (floating point; no default) is the time (in seconds) it takes to switch to the indicated view from a different view, for the follow cam, first person cam, and long cam respectively.
- **sniperZoomTime** (floating point; cannot = 0.0; no default) is the time (in seconds) it takes to switch to zoom view from a different view.
- **sniperFov** (floating point; cannot = 0.0; no default) is how much of the field of view in front of the ship is used for the zoom view. This portion of the FOV is expanded to fill the screen, thereby magnifying that FOV for the player. (The smaller the `sniperFov` value, the greater the magnification)

Example:

```
defres rtAI ( FOLLOW_CAM )
(
   Type.Klown = ( KlownName = "Camera" )

   // normal chase cam view
   AIDataValue ( "folOffsetPosition",  Vector,     ( Val = ( 0.0, -55.0, 20.0 )))
   AIDataValue ( "folLagTime",         Float,      ( Val = 0.1 ))
   AIDataValue ( "folSwitchTime",      Float,      ( Val = 1.0 ))

   // first person view
   AIDataValue ( "fpOffsetPosition",   Vector,     ( Val = ( 0.0, 5.0, 0.0 )))
   AIDataValue ( "fpLagTime",          Float,      ( Val = -0.006 ))
   AIDataValue ( "fpSwitchTime",       Float,      ( Val = 1.0 ))

   // long shot used for death
   AIDataValue ( "longOffsetPosition", Vector,     ( Val = ( 200.0, -1200.0, -200.0 )))
   AIDataValue ( "longLagTime",        Float,      ( Val = 0.01 ))
   AIDataValue ( "longSwitchTime",     Float,      ( Val = 10.00 ))

   AIDataValue ( "sniperFov",          Float,      ( Val = 0.2 ))
   AIDataValue ( "sniperZoomTime",     Float,      ( Val = 1.0 ))
)
```

16.1.2 Free Camera (FreeCam Klown)

- *Klown Code File:* FreeCam.fsm
- *Klown:* FreeCam
  —*Type:* CHFSM
  —*Uses External Klowns:* PlayerInputEngines, PlayerCamera

This klown controls a free floating camera. It is controlled like the player ship but has no model, guns, powerups, or first person view. It also has no AI parameters of its own.

16.2 Entity Klown

- *Klown Code File:* entity.fsm
- *Klown:* Entity
  —*Type:* CHFSM
  —*Uses External Klowns:* Health, Collision, ElectronConductive, DeathPowerUp
  —*Has Internal Klown:* Entity

The Entity klown is used for simple objects that just float in space, take damage, and react to collisions. The klown is used mostly for space debris like ship wreckage. It has no internal AI parameters. Objects that use the Entity klown are automatically assigned `kGarbageType` for sensing purposes.

16.3 Explosion Klown

- *Klown Code File:* explosion.fsm

- *Klown:* Explosion
  —*Type:* HFSM

```
AIDataValue ( "deathSequence",   Resource,   ( Val = ( rtSequence, <resource ID> )))
```

An object assigned the Explosion can play a death sequence when the object is killed:

- **deathSequence** (resource; no default) is the `rtSequence` resource that plays when the object is killed.

In *Sinistar*, only one AI related to the destruction of the jumpgate uses the Explosion klown, and this AI does not assign a death sequence:

```
defres rtAI (BOSS1_RING)
(
   Type.Klown = ( KlownName = "Explosion" )
   AIDataValue ( "bogus", Integer, ( Val = 0 ))
)
```

16.4 Factories

Factories are objects at which other objects can appear (so that it looks like the factory is producing the objects). Two factory klowns are defined in the *Sinistar* code but neither is used by any object in the game. *Note:* Factory klowns have nothing to do with the `rtDatabase (DB_FACTORY )` resources that define the types of objects that can appear in a level!

16.4.1 Factory Klown

- *Klown Code File:* factory.fsm
- *Klown:* Factory
  —*Type:* Submachine

```
AIDataValue ( "FactorySequence",   Resource,   ( Val = ( rtSequence, <resource ID> )))
AIDataValue ( "NumCharges",        Integer,    ( Val = <#> ))
AIDataValue ( "RechargeTime",      Float,      ( Val = <#> ))
```

The Factory klown is a submachine that allows objects to be emitted at the object using the klown. If the factory object has launch pads, the emitted objects appear at those pads (one per pad in the order of the pads); otherwise they appear at the origin point of the factory object.

- **FactorySequence** (resource; no default) is the `rtSequence` resource that plays on the factory. This sequence should emit the object for the factory.
- **NumCharges** (integer; default is -1) is the number of times the factory may operate (play its `FactorySequence`). If not specified or set to 0 or less, the factory operates without limit.
- **RechargeTime** (floating point; default is 10.0) is the amount of time (in seconds) the factory must pause after playing one `FactorySequence` before it plays its next one.

Note: Factory is a submachine and thus is used only when assigned to another klown. See Simple Factories below.

Example:

```
AIDataValue ( "FactorySequence",   Resource,   ( Val = ( rtSequence, EXAMPLE )))
AIDataValue ( "NumCharges",        Integer,    ( Val = 17 ))
AIDataValue ( "RechargeTime",      Float,      ( Val = 30.0 ))
```

16.4.2 SimplyFactory Klown

- *Klown Code File:* factory.fsm
- *Klown:* SimpleFactory
  —*Type:* CHFSM
  —*Uses External Klowns:* Team, Health, Collision, Factory
  —*Has Internal Klown:* SimpleFactoryGuts

The SimpleFactory klown allows an object in the game to be set up as a factory, with team, health, and collision attributes. It uses the Factory submachine (see above) to run the factory. An object using the SimpleFactory klown does not start factory operations until its SimpleFactoryGuts internal klown receives a message to do so.

16.5 Fog Klown

- *Klown Code File:* fog.fsm
- *Klown:* Fog
  —*Type:* HFSM

```
AIDataValue ( "FogDistance",    Float,      ( Val = <#> ))
AIDataValue ( "FogIntensity",   Float,      ( Val = <#> ))
AIDataValue ( "FogRed",         Float,      ( Val = <#> ))
AIDataValue ( "FogGreen",       Float,      ( Val = <#> ))
AIDataValue ( "FogBlue",        Float,      ( Val = <#> ))
```

The Fog klown allows fog effects to be added to levels:

- **`fogDistance`** (floating point; default is 0.0) is the distance (in meters) from the player ship at which the fog appears to be.
- **`fogIntensity`** (floating point; =>0.0 and =< 1.0; default is 1.0) is the intensity of the fog, from 0.0 (invisible) to 1.0 (totally obaque).
- **`fogRed`** (floating point; =>0.0 and =< 1.0; default is 0.0) is the amount of red in the fog, from 0.0 (no red) to 1.0 (maximum red).
- **`fogGreen`** (floating point; =>0.0 and =< 1.0; default is 0.0) is the amount of green in the fog, from 0.0 (no green) to 1.0 (maximum green).
- **`fogBlue`** (floating point; =>0.0 and =< 1.0; default is 0.0) is the amount of blue in the fog, from 0.0 (no blue) to 1.0 (maximum blue).

In *Sinistar*, only one AI related to the destruction of the jumpgate uses the Explosion klown, and this AI does not assign a death sequence:

```
AIDataValue ( "FogDistance",  Float, ( Val = 1400.0))
AIDataValue ( "FogIntensity", Float, ( Val = 0.5   ))
AIDataValue ( "FogRed",       Float, ( Val = 0.489 ))
AIDataValue ( "FogGreen",     Float, ( Val = 0.300 ))
AIDataValue ( "FogBlue",      Float, ( Val = 0.214 ))
```

16.6 Trap Klown

- *Klown Code File:* trap.fsm
- *Klown:* Trap
  —*Type:* CHFSM
  —*Uses External Klown:* Team
  —*Has Internal Klown:* TrapGuts

```
AIDataValue ( "EnemySensor",        Resource,    ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "EnemySequence",      Resource,    ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "EnemyLocSequence",   Resource,    ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "TrapSequence",       Resource,    ( Val = ( rtBodySensor, <resource ID> ))
AIDataValue ( "NumCharges",         Integer,     ( Val = <#> ))
AIDataValue ( "RechargeTime",       Float,       ( Val = <#> ))
```

Note: This klown was originally intended for traps on Levels 5 and 9, but it was not used in the gold version of *Sinistar*.

The TrapGuts klown plays a set sequences (that can have great effects and do nasty things to the player) when triggered. A trap can have a number of charges, and must recharge for a time between being triggered again. Use the following parameters:

- **EnemySensor** (resource; no default) is the `rtBodySensor` resource that the trap uses to sense the presense of an enemy. When an enemy comes with this sensor's range, the trap is triggered and its sequences play.
- **EnemySequence** (resource; no default) is the `rtSequence` resource that plays on the enemy when the trap is triggered.
- **EnemyLocSequence** (resource; no default) is the `rtSequence` resource that plays at the enemy's location when the trap is triggered.
- **TrapSequence** (resource; no default) is the `rtSequence` resource that plays on the trap's object when the trap is triggered.
- **NumCharges** (integer; default is 1) is the number of times the trap may be triggered (play its sequences). Once a trap has used up all its charges, its klown is destroyed.
- **RechargeTime** (floating point; default is 0.0) is the amount of time (in seconds) the trap must recharge after being sprung before it can be triggered again.

Example:

```
AIDataValue ( "EnemySensor",        Resource,    ( Val = ( rtBodySensor, EX_GET_TIMMY ))
AIDataValue ( "EnemySequence",      Resource,    ( Val = ( rtBodySensor, EX_SMACK_TIMMY ))
AIDataValue ( "EnemyLocSequence",   Resource,    ( Val = ( rtBodySensor, EX_EXPLOSION ))
AIDataValue ( "TrapSequence",       Resource,    ( Val = ( rtBodySensor, EX_TRAP_IN_ACTION
                ))
AIDataValue ( "NumCharges",         Integer,     ( Val = 3 ))
AIDataValue ( "RechargeTime",       Float,       ( Val = 12.0 ))
```

17 Administrative FSMs

17.1 CzKlownz FSM

- *Klown Code File:* CzKlownz.fsm
- *Klown:* (none)

This is an empty file. It exists so that the FSM compiler can create automatically CzKlownz_FSM.cpp and CzKlownz_FSM.h files. CzKlownz_FSM.cpp registers klown names when they are created in the game. It returns an error message if the game tries to create a klown that already exists with a given name or ID.

17.2 ID FSM

- *Klown Code File:* id.fsm
- *Klown:* (none)

This is the message map file for the FSM compiler. This file is generated automatically by the FSM compiler to keep track of the message types being used by the klowns.

18 Appendices

18.1 Appendix: Some Useful Math

18.1.1 Approximate Value of Pi

pi = 3.141592...

18.1.2 Degrees vs. Radians

radians = degrees x pi / 180

degrees = radians x 180 / pi

18.1.3 Quick Table of Cosines

Angle	*Cosine*
0°	1.000
5°	0.9962
10°	0.9848
15°	0.9659
20°	0.9397
25°	0.9063
30°	0.8660
45°	0.7071
60°	0.500
90°	0.0000
120°	-0.5000
135°	-0.7071
150°	-0.8660
180°	-1.0000
210°	-0.8660
225°	-0.7071
240°	-0.500
270°	0.0000
300°	0.5000
315°	0.7071
330°	0.8660
360°	1.0000

18.2 Appendix: Summary of Difficulty Effects

18.2.1 Difficulty and Ignoring Difficulty

- *Klown Code File:* Difficulty.fsm
- *Klown:* Difficulty

```
AIDataValue ( "IgnoreDifficulty",   Integer,    ( Val = ( <#> ))
```

The difficulty level in the game's code can range from 0.0 (easiest) to 1.0 (hardest). The player can set the game to play at one of four of these levels:

- Easy difficulty level is set to 0.1.
- Normal difficulty level is set to 0.5.
- Hard difficulty level is set to 0.8.
- Sinistar difficulty level is set to 1.0.

Difficulty affects many aspects of the play of the game, including the crystals available on asteroids, the capabilities of the player ship, and the abilities of enemy ships (including bosses). Some elements relating to difficulty are tunable through AI parameters, and these are covered below.

Other elements are hardcoded. However, some of these can be circumvented by setting the following parameter in the AI of various objects:

- **`IgnoreDifficulty`** (integer; default is 0) specifies whether (`Val = 1`) or not (`Val = 0`) the game ignores the player-set difficulty level and uses the hardest (1.0) level instead.

`IgnoreDifficulty` has effect in any klown that calls on the Difficulty klown. These are summarized in the following sections.

Example:

```
defres rtAI ( TICK1z )
(
   ...
   AIDataValue ( "IgnoreDifficulty",   Integer,    ( Val = ( 1 ))
)
```

In the above example, an ship that uses the `TICK1z` AI will always use the 1.0 difficulty level for its health and movement abilities.

18.2.1.1 Health and ExcAlive Klowns

- *Klown Code Files:* health.fsm, excalive.fsm
- *Klowns:* Health, ExcAlive

Any object that uses Health or ExcAlive can have its starting/maximum possible health adjusted by difficulty level. (Enemy ships, turrets, shots, etc., use Health while bosses use ExcAlive; in both cases, however, the following parameters are named and work the same.)

Use `minHealth` and `maxHealth` (see Health in Common Ship Parameters and Boss HMO in Common Boss Parameters) to tune this. For example, to have a boss's health start low in low difficulty levels and be scaled to higher values in higher difficulty levels, set its `minHealth` low and its `maxHealth` high. The Health klown calls on the

Difficulty klown, and thus the `IgnoreDifficulty` parameter can affect your `minHealth` and `maxHealth` settings.

18.2.1.2 AutoHealth Klown

- *Klown Code File:* health.fsm
- *Klowns:* AutoHealth

Any object (so far, just the player ship) that uses AutoHealth (and can spend crystals) can have the amount of health it gains from crystal expenditure adjusted by difficulty level. Use `easyHealthPerSec`, `hardHealthPerSec`, `easyCrystalPerSec`, and `hardCrystalPerSec` (see The Player Ship) to tune this. Unlike the Health klown, the AutoHealth klown does not calls on the Difficulty klown, and thus the `IgnoreDifficulty` parameter has no effect on autohealth.

18.2.1.3 Engine Klowns

- *Klown Code File:* ship.fsm
- *Klowns:* various, see table below

Engine Klown	*Also uses these engines*	*Uses `IgnoreDifficulty`*	*Parameters affected by Difficulty Level*
MatchEngines (for inertial movement)	—	Yes	`normalMotion` `normalControl`
MatchEnginesInstantly (for inertialess movement)	—	Yes	`normalMotion` only
OrientEnginesInternal (for turret-like orientation)	—	No	`normalMotion` (used just for orientation)
PlayerInputEngines	MatchEngines	via MatchEngines	via MatchEngines
MoveEngines	MatchEngines	via MatchEngines	via MatchEngines
MoveEnginesNoSpin (for inertial movement without spin)	MatchEngines	via MatchEngines	via MatchEngines
MoveEnginesWithLat	MatchEngines	via MatchEngines	via MatchEngines
OrientEngines (for inertially moving and orienting)	MatchEngines and OrientEnginesInternal	via MatchEngines	via MatchEngines and OrientEnginesInternal
AccurateOrientEngines (for inertial movement and infinitely accurate orientation)	MatchEnginesInstantly and OrientEnginesInternal	via MatchEnginesInstantly	via MatchEnginesInstantly and OrientEnginesInternal
MissileEngines (for missile-like movement, "whatever that is")	MatchEngines	via MatchEngines	via MatchEngines
SRMissileEngines (for inertialess movement to make short-range missiles more accurate)	MatchEnginesInstantly	via MatchEnginesInstantly	via MatchEnginesInstantly
SpinEngines	MatchEngines	via MatchEngines	via MatchEngines
CircleEngines	—	No	—

(for circling movement)			
CircleEnginesNoSpin (for inertialess circling)	—	No	`circleDistance`
BarrelRollEngines (for barrel rolls)	—	No	`barrelSpinEffort` `barrelStrafeEffort`
RollTowardEngines (to roll ships with "battleship broadsides" weapons)	—	No	`rollEffort`
OrbitalEngines (for orbitals)	—	No	—

Almost all engines used to laterally move or orient an object (like ships and turrets) are affected by difficulty level, with the best movement and orientation abilities occurring at the highest difficulty level. Note that almost anything that moves or orients is affected (ships, bosses, turrets, missiles, mines, etc.), and thus your `normalMotion`, `normalControl`, and other settings can be affected (see the above table). Engines that directly or indirectly call on the Difficulty klown also have their settings affected by the `IgnoreDifficulty` parameter.

`circleDistance` used for circling by objects with decoupled spins (i.e., "UFO" style movement) are affected by difficulty level and by the `IgnoreDifficulty` parameter. `circleDistance` used for circling by other objects, however, are not affected by difficulty level.

Orbital engines (used by orbitals) are not affected by difficulty level.

`normalMotion` and `normalControl` are affected by difficult level as follows: The settings in their `rtBodyMotion` and `rtBodyControl` resources are used as specified on the highest difficulty level. A factor in the code (one factor each for `normalMotion` and `normalControl`) scale these settings evenly across levels. Current, each factor is set to scale the settings so that on difficulty level 0.0 they are at 50% their difficulty level 1.0 values. (While these factors are hardcoded, it should be fairly easy for a programmer to change their values.)

18.2.1.4 Avoidance Klown

- *Klown Code File:* avoid.fsm
- *Klown:* Avoidance

Objects that use Avoidance has their avoidance abilities based on difficulty level, with the best avoidance occurring at the highest difficulty level. Since avoidance uses engines, it can be affected by the `IgnoreDifficulty` parameter.

There are three avoidance behaviors that use difficulty level settings:

- +Y avoidance is used at or above 0.8 difficulty level (i.e., at Hard and Sinistar levels). +Y avoidance means the object tries to maneuver out of the way of detected target's +Y vector. (That is, the avoidance doesn't assume the target is stationary but checks to see if it is moving toward the object, too.)
- Shot avoidance is used at or above 0.4 difficulty level (i.e., at Normal and above levels). Shot avoidance means the object tries to maneuver out of the way of detected shots.
- Missile avoidance is used at or above 0.3 difficulty level (i.e., at Normal and above levels). Missile avoidance means the object tries to maneuver out of the way of detected missiles.

The difficulty levels for these behaviors are hardcoded in avoid.h, but it should be relatively easy for a programmer to change them if needed.

18.2.1.5 Roid Klown

- *Klown Code File:* roid.fsm
- *Klown:* Roid

Asteroids (objects that use the Roid klown) can have the number of crystals they emit adjusted by difficulty level. Use `minDamagePerCrystal` and `maxDamagePerCrystal` (see Asteroids) to tune this.

18.2.1.6 Aiming Shots (Leading)

- *Code File:* motion.cpp
- *Klown:* none

When an object aims to fire a shot at a target, it leads its target if the difficulty level is at or above 0.3. (This level is hardcoded but should be relatively easy for a programmer to change if needed.)

Also, when the final target-leading offset is calculated for the shot, the difficulty level affects the accuracy of the offset, with the lower difficulty levels having less accurate offsets.

18.2.1.7 Player Ship Auto-Aiming

- *Code File:* weapon.cpp
- *Klown:* none

When playing on the Easy difficulty level (only) the player ship has auto-aiming, which can affect (only) the mining laser, the concussion shot, and the mind control shot. See Auto-Aiming (for Easy Difficulty) in The Player Ship for details.

18.2.1.8 Player Ship Turbo Crystal Magnet

When playing on the Easy difficulty level (only) the player ship has a souped-up crystal magnet. See Turbocharged Crystal Magnet (for Easy Difficulty) in The Player Ship for details

18.2.2 Sequences Tagged to Difficulty Level

- *Klown Code File:* Difficulty.fsm
- *Klown:* DifficultySequences

```
AIDataValue ( "easySequence",       Resource,   ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "mediumSequence",     Resource,   ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "hardSequence",       Resource,   ( Val = ( rtSequence, <resource ID> ))
AIDataValue ( "sinistarSequence",   Resource,   ( Val = ( rtSequence, <resource ID> ))
```

You can specify different sequences for each difficulty level in the AI for a level (in the `defres rtAI ( MISSION_SETUP )` resource in level_X.r files). When the level runs, its appropriate difficulty level sequence plays, in addition to any other sequence that normally would play. For example, if the game is being played on Normal difficulty, then when a level runs its `mediumSequence` sequence plays, in addition to any other sequence that normally would play.

At most, one *difficulty level* sequence will play. (For example, if the game is being played on Normal difficulty, a level's `mediumSequence` plays but not its `easySequence`, `hardSequence`, or `sinistarSequence`.) If no difficulty level sequence is specified for a particular difficulty level, then no difficulty level sequence plays. (For

example, if you did not specify a `mediumSequence` for a level, then no difficult level sequence plays when you play the level on Normal difficulty.)

The parameters controlling difficulty level sequences are:

- **easySequence** (resource; no default) specifies the sequence that plays when the level is played on Easy difficulty.
- **mediumSequence** (resource; no default) specifies the sequence that plays when the level is played on Normal difficulty.
- **hardSequence** (resource; no default) specifies the sequence that plays when the level is played on Hard difficulty.
- **sinistarSequence** (resource; no default) specifies the sequence that plays when the level is played on Sinistar difficulty.

Note: You can specify difficulty level sequences in any AI that uses the Mission klown (contains `Type.Klown = ( KlownName = "Mission" )` in its AI resource), although in *Sinistar* only levels currently use the Mission klown.

Example:

```
defres rtAI ( MISSION_SETUP )
(
   Type.Klown = ( KlownName = "Mission" )
   ...
   AIDataValue ( "startSequence",    Resource, ( Val = ( rtSequence, MISSION_SETUP )))
   AIDataValue ( "easySequence",     Resource, ( Val = ( rtSequence, LVL01_SEQ_EASY )))
   AIDataValue ( "mediumSequence",   Resource, ( Val = ( rtSequence, LVL01_SEQ_NORMAL )))
   AIDataValue ( "hardSequence",     Resource, ( Val = ( rtSequence, LVL01_SEQ_HARD )))
   AIDataValue ( "sinistarSequence", Resource, ( Val = ( rtSequence, LVL01_SEQ_SINISTAR
                 )))
   ...
)

defres rtSequence ( MISSION_SETUP )
(
   ...               // Emit all sorts of stuff at all sorts of time, just like you do now
)

defres rtSequence ( LVL01_SEQ_EASY )
(
   ...               // Emit stuff when level played on Easy
)

defres rtSequence ( LVL01_SEQ_NORMAL )
(
   ...               // Emit stuff when level played on Normal
)

defres rtSequence ( LVL01_SEQ_HARD )
(
   ...               // Emit stuff when level played on Hard
)

defres rtSequence ( LVL01_SEQ_SINISTAR )
(
   ...               // Emit stuff when level played on Sinistar
```

)

In the above example, suppose the player is playing at Hard difficulty. When the player enters the level, its starting sequence (triggered by `startSequence`) and hard difficulty level sequence (triggered by `hardSequence`) play.

19 Index

The index lists all the `AIDataValue` names and AI-related macros that are documented in this document. Note that some names appear more than once on the list; these typically occur when different klowns happen to use the same name and are not necessarily the same. (Some might indeed be the same; I tried to catch all the ones in common but I might have missed some.)