VSG v4.2 User's Guide

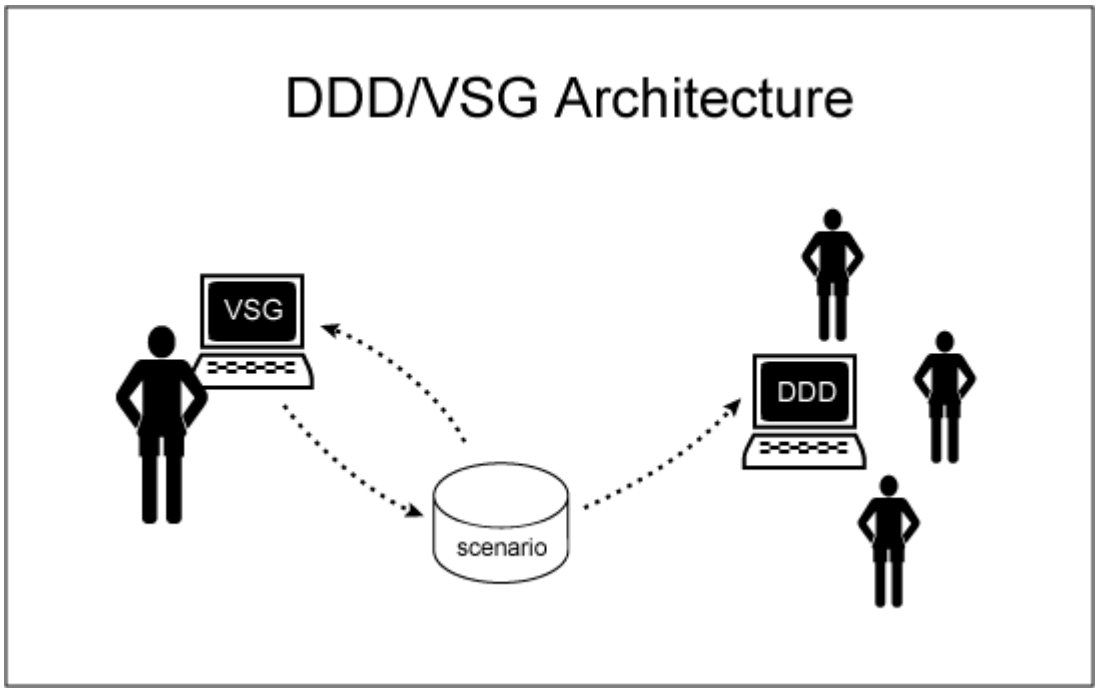
November 2011

Contents

1.0 Introduction

The Visual Scenario Generator (VSG) is an interactive visual editing environment for the creation and modification of Aptima's Distributed Dynamic Decision-making system (DDD) scenarios. The DDD runs simulations (also known as *experiments* or *training sessions*) for individuals or teams participating in pre-defined "scripted" scenarios. Subjects will participate in a simulation or students will use the DDD for training.

The scenario writer uses VSG to script a new scenario and generate a scenario file that will be loaded onto the DDD Simulation Server and then used by the DDD client software, as illustrated below. The scenario writer also can use the VSG to modify an existing scenario file.



This document describes how to use VSG to create scenarios:

1. Scenario Overview
2. How to start the VSG
3. The basic workflow when creating a scenario
4. The basic steps involved in creating a scenario
5. Advanced topics
6. Examples of more complex components of scenarios

2.0 Scenario Overview

2.1 What is a Scenario?

Players who participate in the experiment (meaning, play the scenario) are referred to as "Decision Makers." They use DDD client software to participate in simulations. Decision Makers are like game players and the simulation is the game: Decision Makers have roles in the simulation. They own and control resources (such as tanks or planes). Decision Makers usually have one or more goals to accomplish during the course of a simulation.

The scenario is a text file that defines the simulation and provides everything the Decision Makers need in the simulated world. Writing a scenario is like creating a puzzle: scenario writers use the VSG first to create all the puzzle pieces needed by the scenario and then use VSG to put those pieces together into a scenario that is ready to use. Finally, the scenario writer will use the VSG to generate a scenario file that will be used by the experimenter.

The VSG gives scenario writers a graphic and easy way to create scenarios from scratch or modify existing scenarios. The VSG also acts to "validate" scenarios. It will only read (and will only write) valid scenarios, as well as allowing scenario writers to work on scenarios as "works in progress" and complete them later.

2.2 The Basic Elements of a Scenario

The following are the most basic elements of a scenario.

- A *Playfield* is where the action takes place during the simulation.
- *Decision Makers* represent players, who are organized into *teams*.
- *Resources* are objects controlled by Decision Makers or by the scenario itself.
- *Events* are actions generated by Decision Makers or by the scenario itself.
- *Networks* are organizations that let Decision Makers share information with each other.

The above list is a very simple overview. This guide describes the elements of a scenario in detail.

To give an example of the basic elements, suppose there is a need for the following scenario. The scenario writer creates two teams: the Red team and the Blue team. The scenario writer also defines Decision Makers, and each Decision Maker is a member of the Blue or Red team. Each team has a plane, so there is a Red plane and a Blue plane. Each plane has a weapon that is 70% effective within a range of two kilometers. When the simulation begins, the Red plane must be flying between two cities (Pyongyang and Seoul) over the Korean Peninsula, and the Blue plane will be on the ground at Busan. The goal of the Blue team is to fly the Blue plane out of Busan, intercept the Red plane, and engage in battle.

The following are the most basic elements that the scenario writer must script for this example:

- The playfield is a map of the Korean Peninsula.
- The Decision Makers are Blue team players and Red team players.
- The resources are the Blue plane, the Red plane, and the weapons located on each plane.
- The events are the actions that establish where each plane is located at the beginning of the simulation and how those planes can be controlled. If the Red plane is controlled by the scenario (meaning, there are no human Red players), then it is necessary to script all of the actions and reactions of the Red plane.
- If there are multiple players on a team or teams, then a network for the Blue team players would give them a way to share sensor information during the simulation. If there are Red players, a network can be established for them, too.

2.3 Understanding a Scenario Before Using VSG to Script It

The scenario writer should have a good understanding of the scenario design before attempting to script it with VSG. In other words, it is a good practice for the scenario writer to know everything needed by the scenario and how the scenario will work when it is used in a simulation. The scenario writer uses VSG as a scripting tool to translate what the writer already knows into a working scenario.

Before using the VSG to create a scenario, the scenario writer should answer some basic questions, including the following:

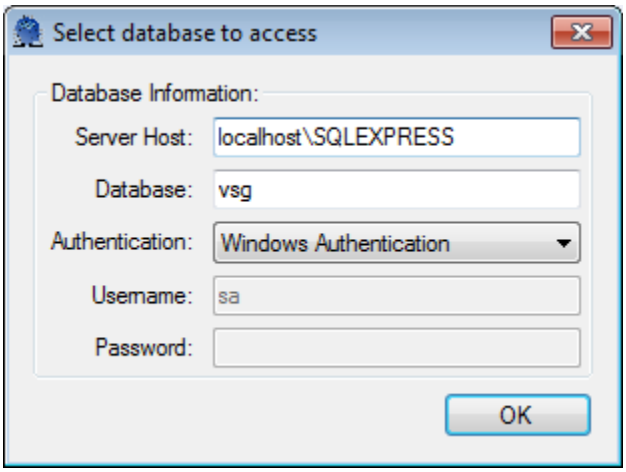
- What is the goal of having the participants use the DDD?
- Will the scenario be used to run an experiment?
- Will the scenario be used to create a training system?
- If the scenario is meant to gather data, what exactly will be measured?
- What player interactions should take place during the experiment?
- What needs to be in place to allow those interactions?

The answers to these and related planning questions will help the scenario writer decide how to set up the teams, players, roles, resources, units (assets), and script the behaviors of those units.

3.0 Starting VSG

Use the following steps to start the VSG:

1. Double-click on the VSG 4.2 desktop icon. Alternatively, use the Microsoft Start menu to select Programs, Aptima, VSG 4.2, and VSG 4.2. The Select Database to Access dialog box is displayed, as shown below.

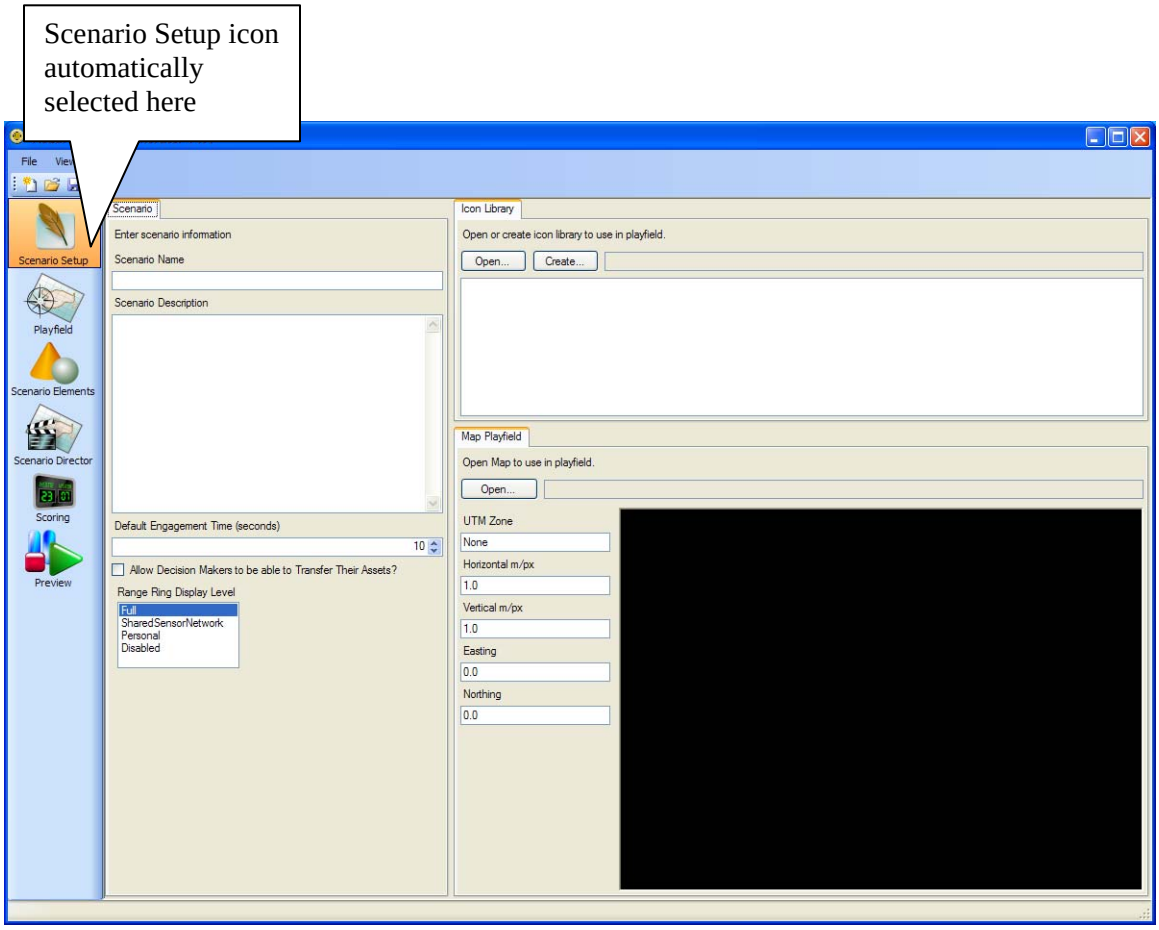


   Note: VSG requires a database that is created as part of the VSG installation process. To use VSG, it is first necessary to connect to this database using the information defined when the database was created, usually during the Installation process.

   Note: Once a new scenario is created, it can be edited. Also, DDD customers who previously wrote their own "hand-made" XML scripts can edit those XML scripts in VSG.

2. The values for Server Host, Database, and Username are displayed by default. If necessary, change the default values for the following:
   - The Server Host box requires the name of the computer on which the database used by VSG is stored.
   - The Database box requires the name of the database (typically, 'vsg').
   - The Username box requires the SQL Server Administration Account.
3. In the Password box, enter the SQL Server Administrator password.

4. Click **OK**. The Select Database To Access dialog box closes, and the VSG window is displayed. The Scenario Setup view is displayed by default, as shown below.



Note: The six icons on the left side of the VSG window represent six different views. Each view represents a specific step in the process of creating a scenario and, therefore has its own unique set of panes. The purpose and content of those panes changes from view to view. The purpose and contents of those panes can also change within a view, depending on the item currently selected.

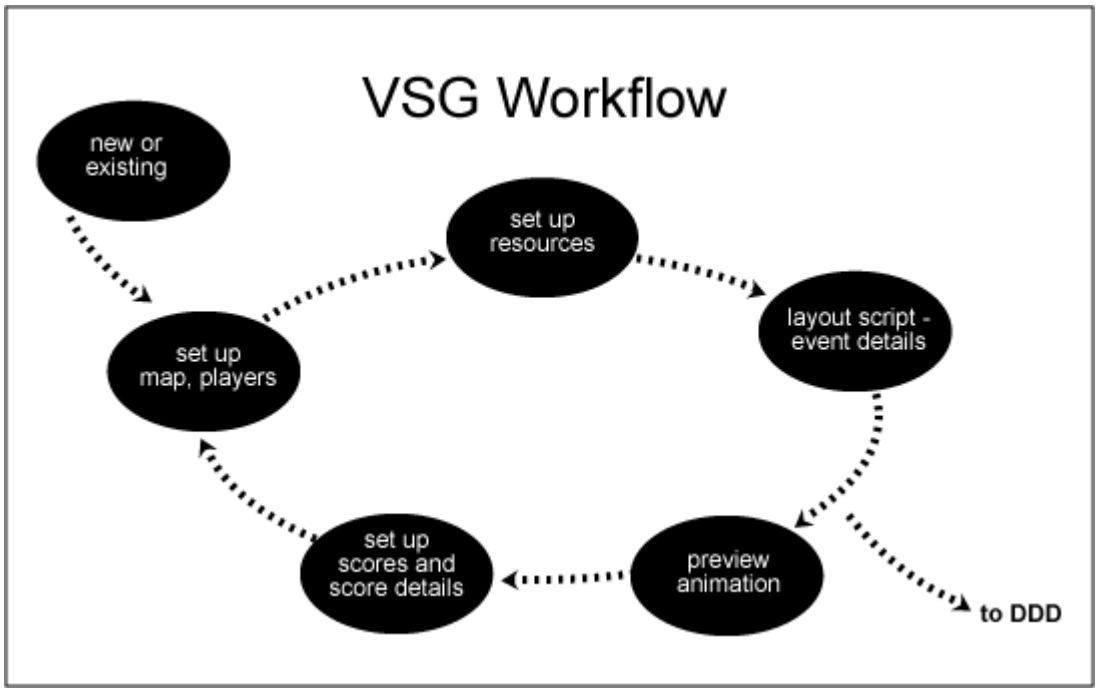
Use the File menu to get started with the following tasks:

- Choose New to create a new database and scenario (default when VSG is initially opened).
- Choose Open to open an existing database and scenario.
- Choose Save As to save the scenario, then choose DDD Scenario File (to save a verified and legal scenario file) or Raw Scenario File (to save work in progress that is not yet verified or legal).

4.0 Overview of the Basic Workflow

The VSG provides six views (representing by the left column of icons on the VSG screen) that guide the scenario writer through the scenario creation process. Writing a scenario is a very

iterative process, as illustrated below. It is a good practice to use the VSG to create and test small pieces of a scenario, then gradually put all the pieces together.



The VSG provides Inline Help that provides definitions, context, and how-to instructions in the Scenario Elements view and the Scoring view. This guide provides instructions for all other views, as well as an overview of using VSG to create a scenario – scenario writers will be directed to use Inline Help where appropriate. The next two sections describe the basic workflow recommended for creating a scenario with VSG.

4.1 Creating the Scenario Pieces (Overview)

The scenario writer's first task is to create the pieces of the scenario:

1. In the Scenario Setup view, establish basic information about the scenario, the playfield, and the icons that will be displayed on the playfield when a simulation is run.
2. In the Playfield view, establish the regions in the playfield and specify their properties.
3. In the Scenario Elements view, define the players, teams, networks, species definitions, and the scenario elements used by the objects that players can move and control on the map.

Creating the Scenario Setup elements and describing the Playfield tend to be simple tasks that can be completed quickly. However, it is not necessary to create all these objects at once. Scenario writers may find it useful to create a few objects and then continue with the next task of putting the pieces together. These steps are described in more detail in the Creating the Scenario Pieces section (below).

Using the previous example of a scenario in which the Red and Blue planes prepare to engage in battle over the Korean Peninsula, the scenario writer uses the Scenario Setup view to establish basic information, including a playfield (a map of the Korean Peninsula) and choosing icons to represent the Red and Blue planes. Next, the scenario writer uses the Playfield view to establish regions in the Korean Peninsula and set properties on those regions. Finally, the scenario writer uses the Scenario Elements view to define roles for players and assign those players to the Red team or the Blue team, describe types of planes and set parameters for them – these are scenario elements.

4.2 Putting the Scenario Pieces Together (Overview)

The scenario writer's second task is to put the scenario pieces together:

1. In the Scenario Director view, create instances of units that will go on the playfield (using the scenario elements that the scenario writer has defined), allocate those units to players, create global events, create unit-specific events, and assign properties to those events.
2. In the Scoring view, create scoring rules and set parameters for scoring. Because a simulation has Decision Makers with goals, its players often earn or lose points during the simulation and are given a score when the simulation ends. The scenario writer is responsible for setting up the scoring system for each scenario.
3. In the Preview view, use VSG to preview the scenario to find out what it will look like when players play a simulation. The Preview view shows the playfield, the icons, and scripted actions. As the scenario is being developed, it is also a good practice to test it using DDD.

In the Scenario Director view, the Red and Blue planes are created by defining units that are based on Species Definitions. (Inheritance rules are described in Section 6.1, 7.1 Units and Inheritance Rules. An F-16 can inherit basic capabilities from a generic type of Air Vehicle, for example.) The scenario writer also assigns a Red plane to a Red Team Decision Maker and a Blue plane to a Blue Team Decision Maker. The behavior of the Red plane is established by scripting events for the Red plane to follow. Likewise, the starting position and the initial movements of the Blue plane is could be established by creating events for the Blue plane.

In this example, the scenario writer uses the Preview view to make sure the Korean Peninsula is displayed as the playfield, the Red plane and the Blue plane appear where expected, and that these objects behave as expected.

Finally, the scenario writer uses the Scoring view to set up a scoring system for the players. These steps are described in more detail in Section 5.0, Putting the Scenario Pieces Together.

5.0 Creating the Scenario Components

Scenario components can be visual (such as tank on a map) or non-visual (such as player names and teams). Some components can be controlled by players or controlled by the scenario itself.

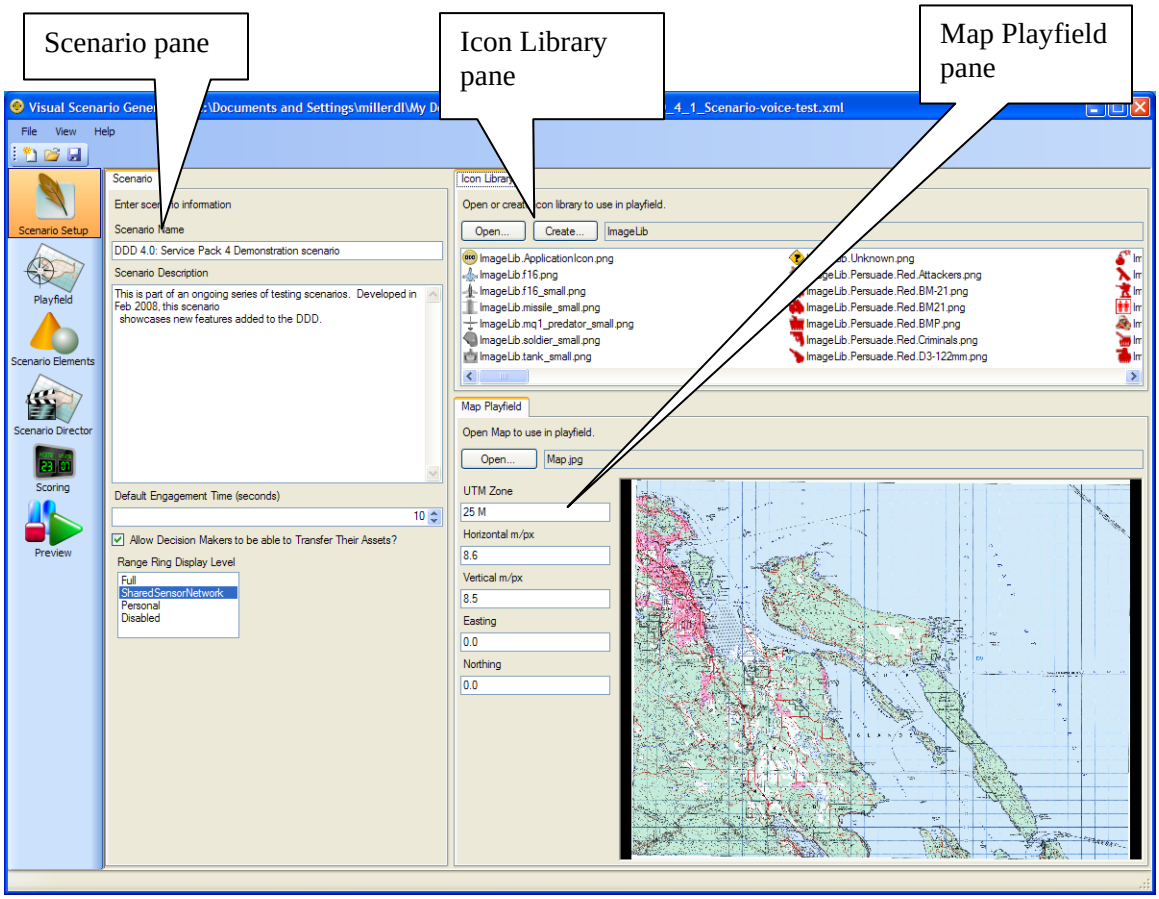
The first step in creating a scenario is to create the scenario components by using the following views in VSG:

- Scenario Setup view
- Playfield view
- Scenario Elements view

The following sections provide an overview of each major view in the VSG and what the scenario writer can accomplish within that view. Although these views are listed in the same order in which they are presented in the VSG, remember that creating a scenario is a fluid and dynamic process (rather than a rigid step-by-step process), and steps can be done in any order.

5.1 Scenario Setup View

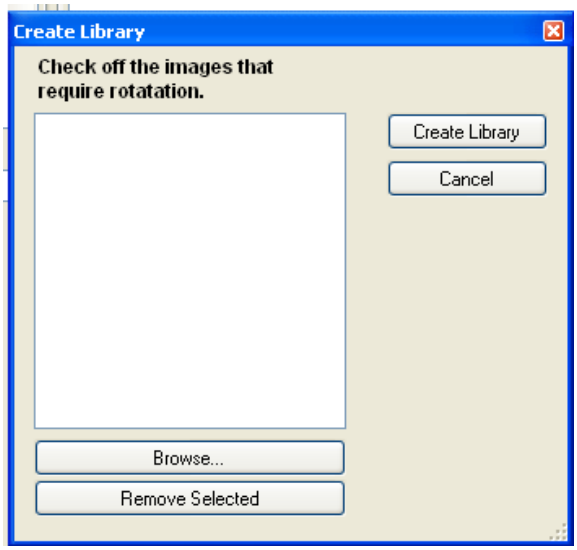
Use the Scenario Setup view to establish basic information about the scenario and the playfield it will use.



Use the Scenario Setup view to perform the following tasks:

- Assign a name to the scenario by entering that name in the Scenario Name box in the Scenario pane.
- (Optional) Write a description of the scenario in the Scenario Description box in the Scenario pane. For example, describe the purpose of the scenario and what players should expect when they play a simulation that uses this scenario.
- Assign a number that represents the default time (in seconds) that any given engagement will take place. Enter a number in the Default Engagement Time box in the Scenario pane. In the above example, the default time is 10 seconds – this means that by default every engagement with another unit on the playfield will last 10 seconds. Depending on the scenario, this could be an attack, a rescue, a drop-off of relief supplies, and so on.
- Check the Transfer of Assets options, if it is desired to allow Decision Makers the capability to Transfer their assets to other players.
- Specify the Range Ring Display Level by selecting a level from the list: Full – everyone can see rings for all players; SharedSensorNetwork – players sharing the same sensor network can see each other's rings; Personal – each player only sees his/her own rings; Disabled – no range rings are displayed.

- Choose the icon library to be used by the scenario in the Icon Library pane. Icons represent units in the playfield, and the scenario writer is responsible for choosing a library of icons and using icons from that library. Aptima provides two icon libraries: the 1477 military standard and the 2525B military standard. Scenario writers can also create their own icon libraries and import them into VSG. Click **Open** to choose an existing icon library (such as one of those provided by Aptima). Alternatively, click **Create** to display the Create Library dialog box (shown below) to add an icon library to the VSG. Any PNG format can be used. When finished, click **Create Library**. Enter a name for the icon library in the Save As dialog box that is displayed. Click **Save**, and the library will be generated.



- Choose a map to use as the playfield. A map file is a TIFF or JPG file of a map that represents where the scenario takes place. Aptima provides a few map files, but scenario writers are welcome to use other map files. Click **Open** in the Map Playfield pane to display the Open dialog box and select a map file.
- Set parameters for the map field in the Map Playfield pane, as described below.

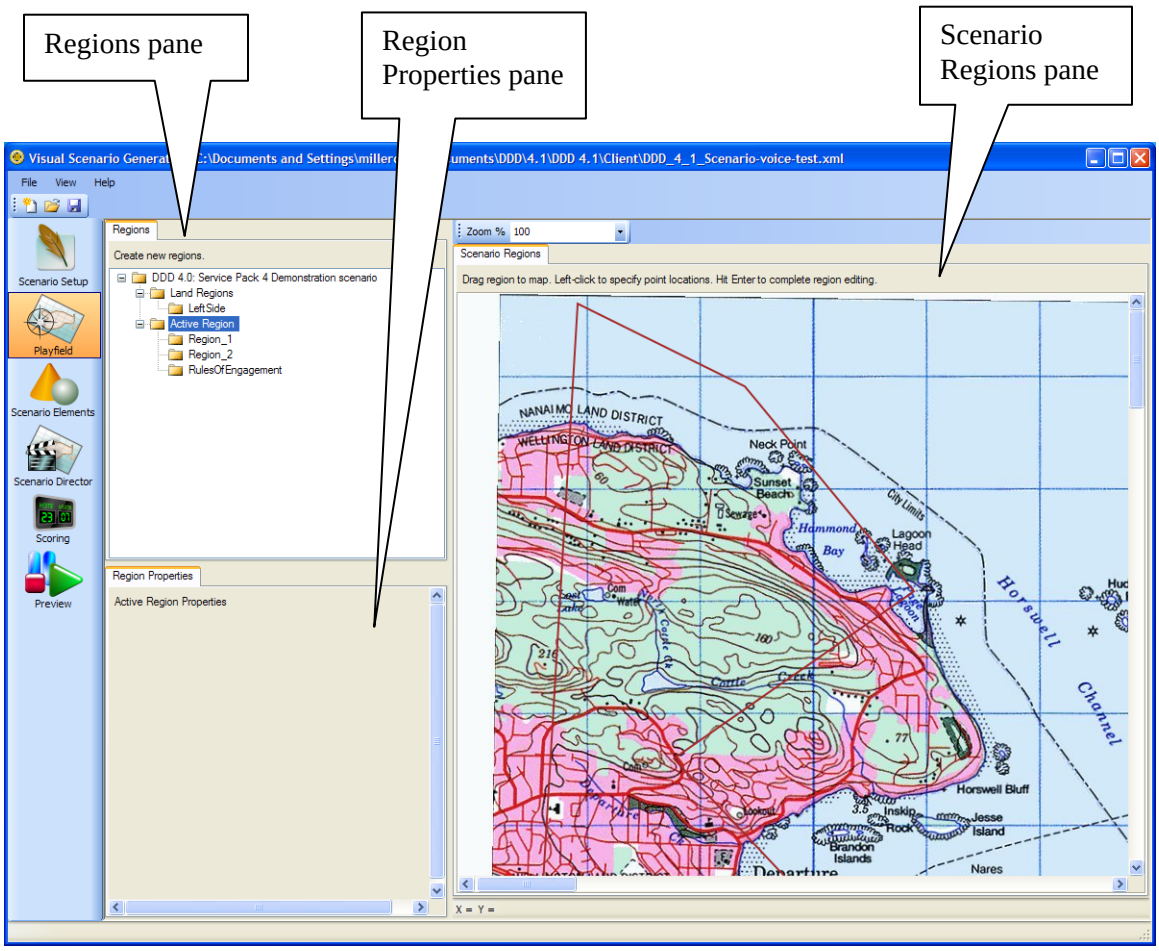
The Universal Transverse Mercator (UTM) Coordinate system provides coordinates on a flat grid of a world map, which is divided into 60 zones. Each zone has a width of six degrees longitude. *Easting* and *northing* are geographic Cartesian coordinates for any given point. Easting is the *x* coordinate (meaning, the measured distance toward the East). Northing is the *y* coordinate (meaning, the measured distance toward the North). In the UTM Coordinate system, northing is the distance to the equator, and easting is the distance to a location in the UTM zone.

Because the map used in a scenario can be a section of a larger map, it is important to specify the origin of the UTM zone as a point on the map that will be displayed as the lower left corner of the screen on which the simulation appears. This location is described in UTM coordinates. In other words, easting and northing are offsets from the origin of the UTM zone.

In the VSG, UTM map parameters include UTM zone, easting, and northing. Also, the *vertical* and *horizontal scaling* values can be set in meters per pixel (m/px). These scaling values make it possible to accurately geographically reference units on the map.

5.2 Playfield View

Use the Playfield view to create regions in the playfield and assign properties to those regions.



In a scenario, there are three basic types of regions: land, sea, and air. Typically, a unit will be able to move in one type of region. For example, a tank can travel across land but cannot fly in the air or sail across the sea. Land units must be restricted to land regions, and sea units must be restricted to sea regions.

Use the Playfield View to perform the following tasks:

- Define land regions (an outlined area of land in which land units can move)
- Edit a land region
- Define active regions (the defined area in which the movement of land units can be affected). For example, a desert or swamp or an urban area might be defined as active regions with the effect of slowing down vehicles trying to move through.

Scenario writers must define at least one land region to establish the playfield. Land units cannot move out of land regions.

5.2.1 Defining a Region

Use the following steps to define a land region:

1. Right click on Land or Active Regions in the tree structure in the Regions pane. The Create Region drop-down menu item is displayed.

2. Click Create Region. The Create Region dialog box is displayed, as shown below.

3. Enter a name for the region in the Name box. Every region must have a name.
4. Enter a description of the region in the Description box.
5. Click **Add**. The name of the new region is now displayed in the tree under Regions section of the tree.
6. IMPORTANT — Click on the new region name in the tree, then drag and drop it onto the Scenario Regions pane to display the "+" cursor. This indicates Draw mode.
7. To draw the outline of the region, drag the cursor and click to establish a point. At least three points must be established. (The land region is a polygon, and its points are connected by straight lines.)
8. Continue dragging the cursor and clicking to establish points until the outline is finished.
9. IMPORTANT — Press Enter to finish.

Typically, the original outline of a region does not have to be exact. Once a region has been created, its outline can be edited.

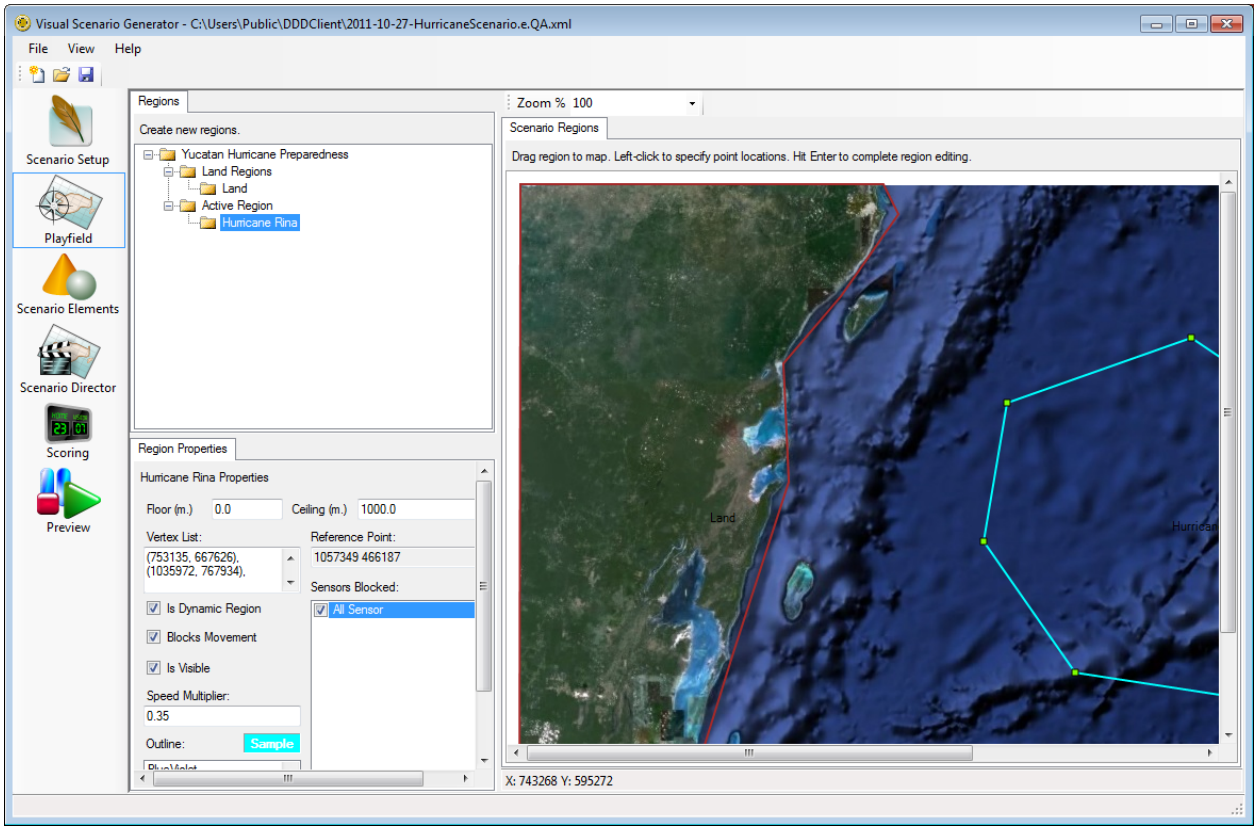
Important: *Active regions should be drawn as convex regions (curving or bulging outward). Concave regions will flag a warning from the VSG, however, the user will be allowed to save the scenario. The resulting scenario will crash if run by the DDD.*

5.2.2 Editing a Land / Active Region

Use the following steps to display a region's outline:

1. If the region is not already displayed in the Scenario Regions pane, click on the name of the region in the tree structure in the Regions pane.
2. Click within the outline of the region. Its points are displayed.

To change the shape of a region's outline, click on a point and drag and drop it to a new location. Alternatively, to specify exact locations, edit the numbers in the Region Properties pane. Using the same map as the previous illustration, the following illustration shows an active region named Hurricane Rina.



Scenario writers who use geo-reference locations should edit the numbers specifying location here.

Use the following steps to add a point to a land region's outline:

1. Position the cursor at the position on the outline where a new point should be added.
2. Right click to display the Add Point Here drop-down menu item. Note: Position the cursor slightly within the land region while keeping it in contact with the land region outline. Otherwise, the land region will be deselected instead of displaying Add Point Here.
3. Click Add Point Here. A new point is added.
4. Click inside the outline to display its points.
5. Drag and drop the new point.

Use the following steps to delete a point from a land region's outline:

1. Right click on the point to be deleted. The Delete This Point drop-down menu item is displayed.
2. Click Delete This Point. The point is removed from the outline.

It is also possible to zoom in and out of the area in which a land region is displayed. To do so, press CTRL and roll the mouse track wheel or use the Zoom % pulldown menu at the top of the map.

5.2.3 Defining an Active Region

An active region represents a three-dimensional space (represented by an extruded polygon with an elevation determined by a floor level and a ceiling level) that has an impact on the way in which the simulation operates. Scenario writers can use active regions to obstruct or reduce the

speed of units traveling through a region. For example, a scenario writer can create an active region that will slow the speed of a tank when it travels through that active region.

Use the following steps to define an active region:

1. Right click on Active Region in the tree structure in the Regions pane. The Create Active Region drop-down menu item is displayed.
2. Click Active Region. The Create Active Region dialog box is displayed.
3. Enter a name for the active region in the Name box.
4. Enter a description of the active region in the Description box.
5. Click **Add**. The name of the new active region is now displayed in the tree structure under Active Regions.

Floor and Ceiling values define the bottom (floor) and the top (ceiling) of the three-dimensional polygon of the active region. These can be left at zero for an active region that is at ground or sea level. Negative values are allowed (such as for an undersea region).

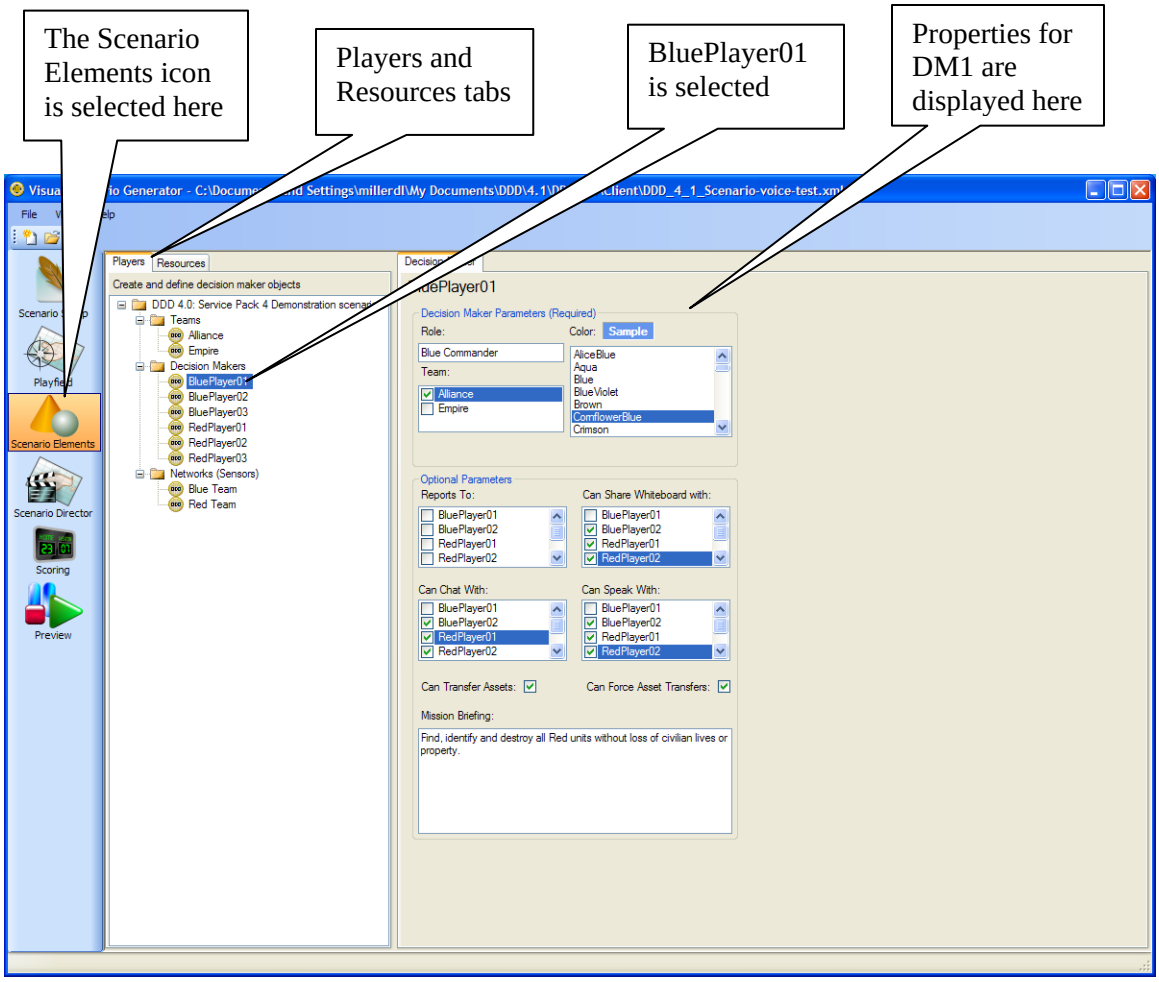
When an active region is selected in the Regions pane, the properties are displayed in the Region Properties pane.

- Vertex List – the region's polygon points can be edited in this field.
- Is Dynamic Region – select this box to specify the region as dynamic (i.e., to specify a moving region, such as a weather pattern, which needs to be "tethered" to a moving object in the scenario).
- Blocks Movement – select this box to prevent units from moving through this region. To allow units to travel through this region, leave the box unchecked.
- Sensor Blocked – to block a unit's sensors (meaning, the unit's ability to sense other units) while in this region, select the box associated with the desired unit(s). To allow sensors to operate normally inside this region, leave the unit's box unchecked.
- Is Visible – this box is selected by default, specifying that the unit is visible when inside this region.
- Speed Multiplier – the speed of any unit inside this region is multiplied by the value specified here. Typically, this value is less than one, which means the unit's speed will be diminished to that value. For example, to slow a unit to half-speed, enter a value of 0.5.
- Outline – the user change the default outline color for the region by selecting another color from the list provided.

Active regions can also be used to have an impact on a player's score. For example, the scenario writer can set up the scoring system so that a player's score is decremented if that player's plane enters an active region that the scenario writer has established as a "no-fly zone." Note: The Examples section of this guide describes how to create a no-fly zone.

5.3 Scenario Elements View

Use the Scenario Elements view to define players, how those players will be organized into teams, and all the resources they will use. The Players tab lets the scenario writer define Decision Makers, Teams, and Networks. The Resources tab lets the scenario writer define all of the resources that will be used by Decision Makers. Both tabs provide extensive Inline Help. Use the Inline Help to gain a better understanding of all scenario elements, their definitions, their uses, and how to define them. The Scenario Elements view provides both context-sensitive help and context-sensitive editing for scenario elements. The name and contents of the right pane will change, depending on what is currently selected in the Players or Resources tab.



Click on each top-level item in the tree structure to display Inline Help about that object. The name of the editing tab (shown above) changes based on the selected element. For example, click on "Decision Makers" to display Inline Help for Decision Makers. On the other hand, when a defined element is selected, its properties are displayed for editing. In the above example, a Decision Maker named BluePlayer01 is selected in the tree, so the editing tab name displays a BluePlayer01 label. The properties of BluePlayer01 are displayed in the editing pane, where they can be edited.

Use the Players tab to define and set values for the following types of objects:

- Teams – players can belong to teams

- Decision Makers – these are roles that players will choose
- Networks – players need networks to share sensor information with each other

Use the Resources tab to define and set values for the following types of objects. Please note that the objects listed below are listed in the same hierarchy as in the Resources tab.

- Engrams – use engrams to create conditional events
- Sensors – units need sensors to detect signals emitted by other units
- Species Definition – a unit is a child object of a species definition
  - States – a unit is always in a state, which can be changed
    - Emitters – units (such as a tank) can emit signals that can be detected by other units' sensors
    - Capabilities – units are given specific abilities that let them engage other units
    - Singleton Vulnerability – how a unit's state will change when affected by a capability
    - Combo Vulnerability – how a unit's state will change when affected by a combination of capabilities

The above definitions are general. See the Inline Help for each of the above objects to learn more about it. The following sections give an overview of the above objects and how they fit together in a scenario.

5.3.1 Overview of Decision Makers, Teams, and Networks

The scenario writer is responsible for defining roles for players to choose (Decision Makers), ways to organize the players (Teams), and ways for those players to share sensor information (Networks). Each player can belong to only one team and Decision Makers on the same team are allies. The scenario writer determines the relationships of teams to each other. To learn more about players and how to define them, click on Decision Makers in the tree to display Inline Help.

It is possible for Teams to engage themselves – this is how the scenario writer can make friendly fire possible. On the other hand, if a team is not specified as being hostile to itself, members of that team will be prevented from engaging (or in a combat simulation, firing) at each other. To learn more about teams and how to define them, click on Teams in the tree to display Inline Help.

Decision Makers can belong to Networks, which are groupings of players that the scenario writer creates as a way for them to share sensor information that their units detect. When playing a simulation in DDD, a player can access all of the sensor information available to any member of any network to which the player belongs. Although it is unlikely that teams that are hostile to each other will share sensor information, but it is possible to create networks with members from more than one team. For example, the scenario writer can define a few teams that cooperate with each other. Any player can belong to more than one network. To learn more about networks and how to define them, click on Networks in the tree to display Inline Help.

5.3.2 Overview of Species Definition

Players require resources, known as *units*. The scenario writer is responsible for defining units for players. However, it is not possible to define a unit without a species definition – this is the starting point. In other words, this is the time to create species definitions. Units will be created and allocated to Decision Makers later in the scenario writing process.

For example, if the scenario requires tanks and planes, create a species definition for types of tanks and a species definition for types of planes. If the scenario requires a total of ten tanks, the scenario writer will use the tank species definition to create all ten tanks in a future step. A

species definition is like a mold that will later be used to stamp out a set of tanks. Likewise, if the scenario requires a total of five planes, the scenario writer will use the plane species definition to create those five planes in a future step. Creating actual units happens when a unit is allocated to a specific Decision Maker.

Every unit is derived from the Species to which it belongs. For example, a species of Air Vehicles might have a sub-species type of unit, such as an F-16 fighter.

Engrams, emitters, sensors, and capabilities are descriptive elements used by individual unit types. VSG makes it possible to define these elements in a way that many units can use them. This gives the scenario writer a very efficient way to define scenario elements once. Later, when the scenario writer defines species and types of units, it is easy to choose a pre-existing element for a unit instead of re-defining the same or similar elements over again for each unit. If slight unit-specific adjustments need to be made, it is possible to edit element properties for specific units.

5.3.3 Overview of Capabilities and Vulnerabilities

VSG gives the scenario writer the ability to define specific capabilities and vulnerabilities for a type of unit. An example of a unit's capabilities is its ability to engage another unit. This could be the capability to attack another unit or to repair another unit, or to interrogate a unit, or to acquire a resource from another unit. An example of that same unit's vulnerability is the way in which it responds and will be affected by the engagement with another unit.

Capabilities and vulnerabilities are about the ability of units to have an impact on each other, and that impact can be positive or negative. A fighter plane might have a laser weapon used to attack another plane. A truck might have the capability to load fuel into a tank. These are capabilities. When a unit has a capability, it has the ability to change another unit's state.

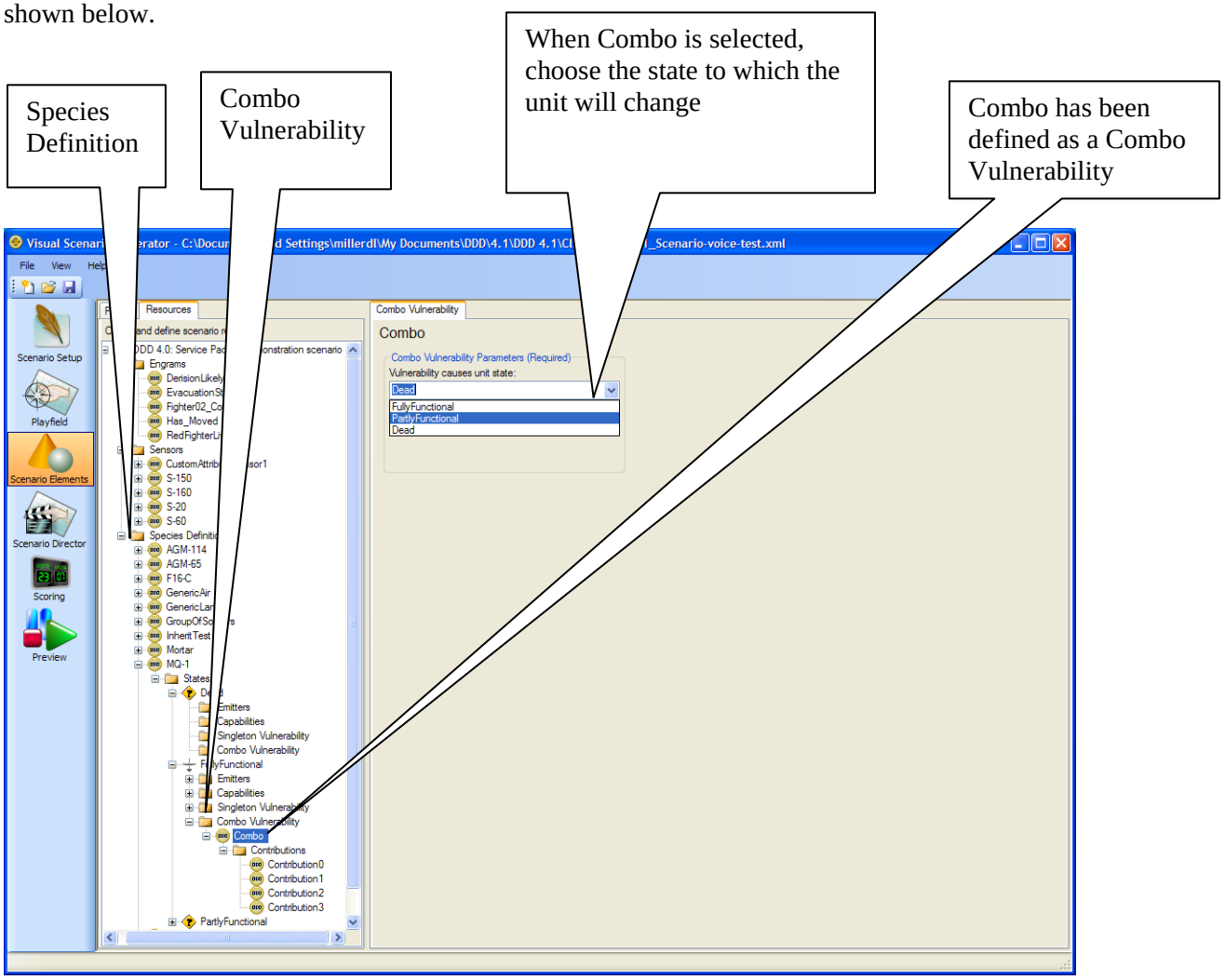
A unit's vulnerability describes how its state will change when that unit is affected by another unit's capabilities. A unit can have many vulnerabilities, and each vulnerability can be associated with a different capability. The scenario writer can modify vulnerabilities by specifying probabilities and ranges, which is detailed in the Inline Help.

As previously mentioned, there are two types of vulnerabilities:

- Singleton Vulnerability – how a unit's state will change when affected by a specific capability
- Combo Vulnerability – how a unit's state will change when affected by a specific combination of capabilities

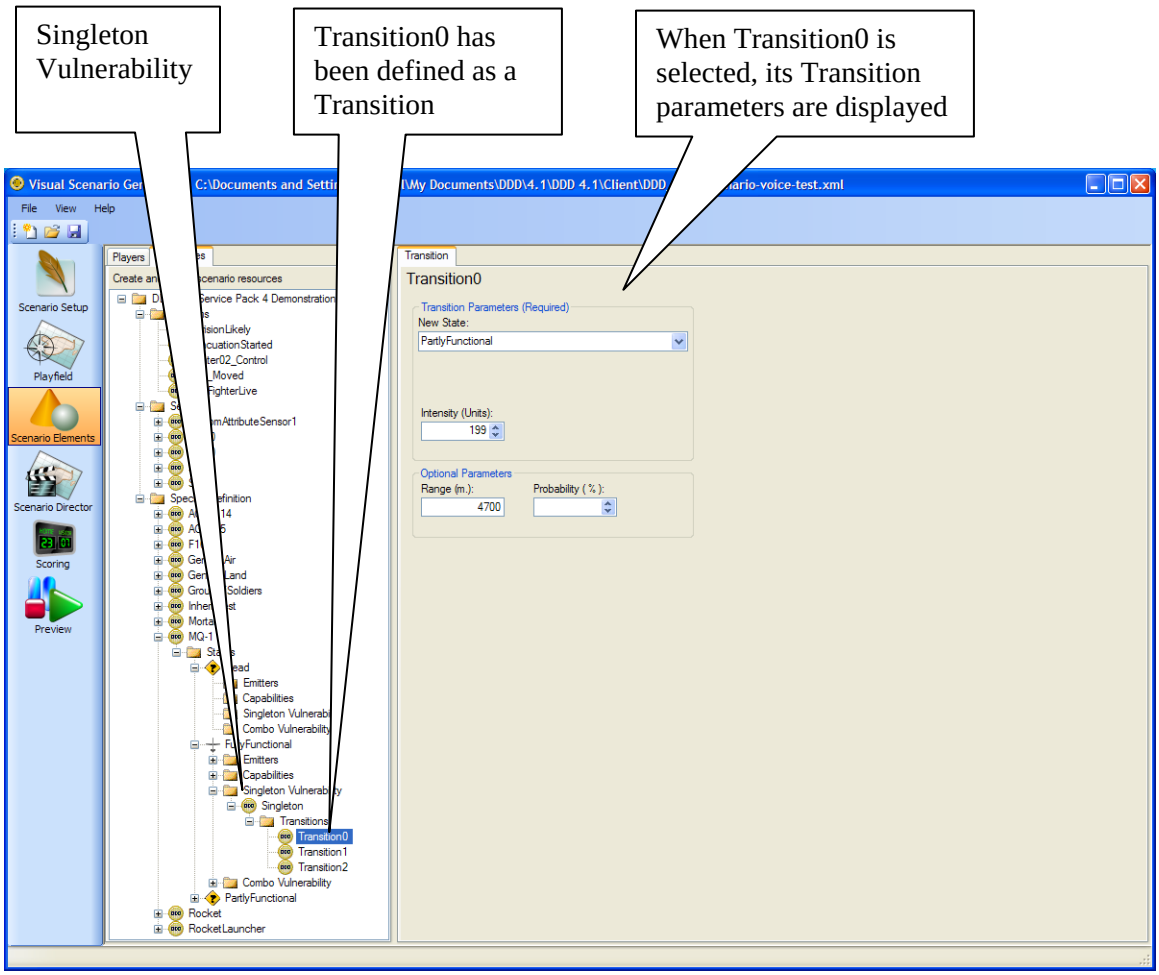
A unit can have many singleton and combo vulnerabilities or none. Note that Capabilities is displayed in the tree structure. Click on Capabilities to display Inline Help to learn more about them and how to define them.

Vulnerabilities are displayed under a unit's States, which in turn is part of a Species Definition, as shown below.



Combo Vulnerabilities use Contributions, which the scenario writer must also define. Notice the Contributions node below Combo in the tree structure above. In VSG, click on Contributions in the tree structure to learn more about Contributions and click on Combo Vulnerability to learn more about Vulnerabilities.

Along the same lines, Singleton Vulnerabilities use Transitions, which the scenario writer must define. Use a Transition to specify the requirements that must be met before a unit will make the transition from one state to another. Notice the Transition node located below Singleton Vulnerability in the tree structure shown below. Click on Singleton Vulnerabilities or Transitions in the tree structure to display Inline Help.



5.3.4 Overview of Engrams

Engrams are global variables that can be tested in conditional events. Engrams can be defined to control the behavior of units or have an impact on the scenario.

For example, suppose riot police will move from one location to another if a city's population begins to riot. This is a conditional event. The scenario writer can create a City Mood engram with the values of Content, Angry, and Rioting. Suppose the default state of the City Mood is Content. If other events that happen during the course of the simulation cause the City Mood to change to Rioting, this will cause the riot police to change their location.

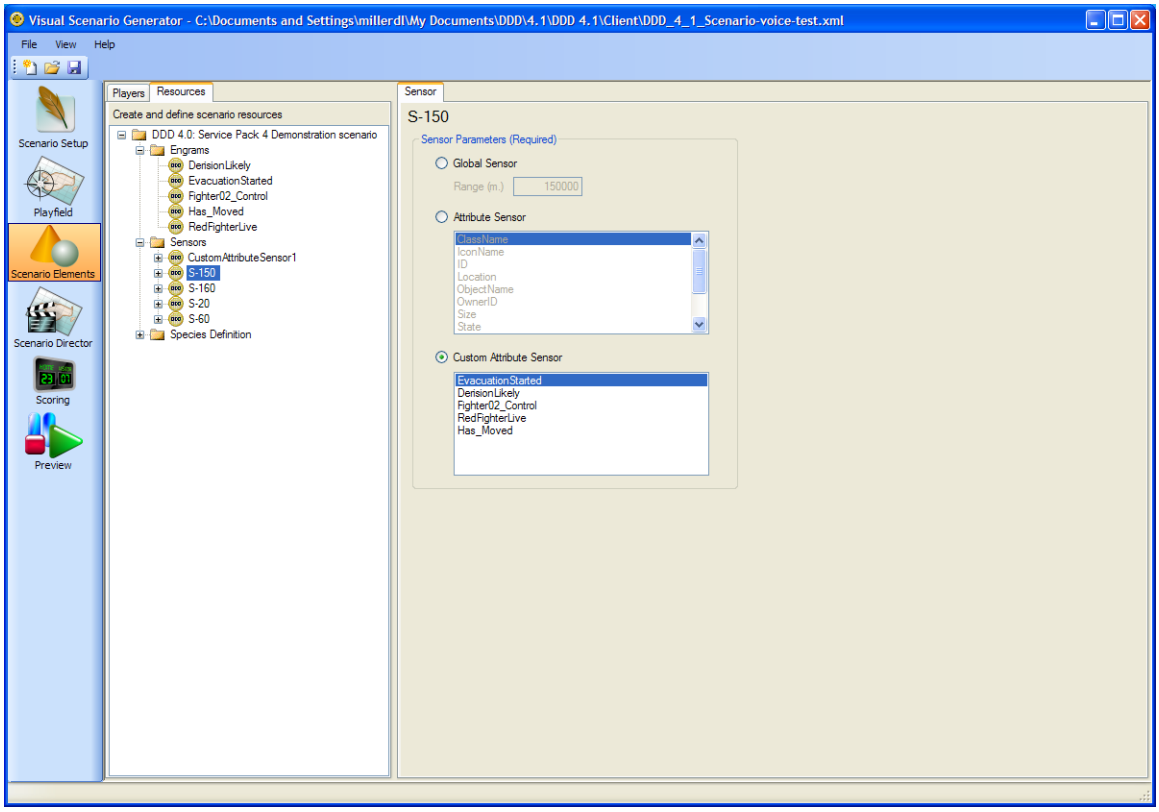
Click on Engrams in the tree structure to display Inline Help, which describes engrams in more detail and provides instructions for defining engrams.

5.3.5 Overview of Sensors and Emitters

Units have the ability to emit signals and to detect signals that are emitted by other units. This means that each unit can have at least one sensor (to detect signals emitted by others) and an emitter (to emit its own signals). The signals emitted by a unit make it possible to identify that unit's attributes. But in order to be detected, a signal first has to be emitted. As previously mentioned, the scenario writer can create networks that allow Decision Makers to access information detected by unit sensors, and Decision Makers can belong to overlapping sensor networks. If two units (for example, Tank1 and Tank2) belong to the same network, then each unit has access to the sensor information collected by the other unit, as well as its own. For example, Tank1 has access to the sensor information it collects and access to the sensor information collected by Tank2.

Once a sensor is defined, each sensor detects only one kind of signal. These are called *attribute sensors*. However, it is possible to define a *global sensor* that will detect all signals.

The DDD has the ability to detect and emit signals from user defined custom attributes. The DDD utilizes Engrams as Custom Attributes. To define a DDD User Defined Custom Attribute from within the VSG, simply define an Engram. (See the previous topic Overview of Engrams). Select the newly created Engram from the Sensor/Emitter definition screen(s). An example is shown in the following VSG screen capture:



Each sensor has a range. This means the sensor has the ability to detect signals within its own range but cannot detect anything outside its range. Sensors also have qualities. For example, a scenario writer can edit a sensor so that its ability to detect a signal weakens at a certain distance or is masked by noise.

For emitters, a scenario writer can specify the level of how well the sensor can work within that range. By editing a level's properties, the scenario writer can also specify the accuracy with

which the signal can be detected. The scenario writer can specify an emitter to be "invisible" by selecting that list item from the list of "Emits a standard attribute", thus making the associated object invisible on the playfield. This option is useful when creating an object to be associated with a dynamic active region (i.e., if the scenario writer does not want DDD players to see the object enabling the region's movement).

To display the Inline Help for sensors, select Sensors in the tree in the Scenario Elements pane. Likewise, to display the Inline Help for emitters, select Emitters.

5.3.6 Overview of States

Every unit has more than one possible *state*, which defines that unit's ability at any given time. A unit is always in one of its states. By default, a unit has two possible states: Fully Functional and Dead. The scenario writer can define new states as part of a Species Definition. When a unit's state changes, it is possible for that unit's icon, sensing abilities, and emitted signals to change.

A unit's current state determines its capabilities, vulnerabilities, sensors, emitters, and parameter values. For example, a unit in its Fully Functional state has all of its capabilities. However, a scenario writer can define a Partially Functional state in which the unit will lose some of its capabilities. On the other hand, the scenario writer can define another state in which the unit will gain capabilities. For example, a tank may re-fuel, which could move it from a Depleted state to a Fully Functional state. Along the same lines, in a different state, a unit's vulnerabilities may increase or that unit may gain new vulnerabilities. The scenario writer can also set optional parameters, for example, that make it possible for a unit to be stolen.

In summary, use states to change a unit's abilities during the course of a simulation. For details, select States (under Species Definitions) in the tree in the Scenario Elements pane to display Inline Help.

After some scenario elements have been defined, the next step is to put them together, as described next. Remember that VSG is flexible, and scenario writers can work however they wish, for example, by working on small chunks of a scenario at a time.

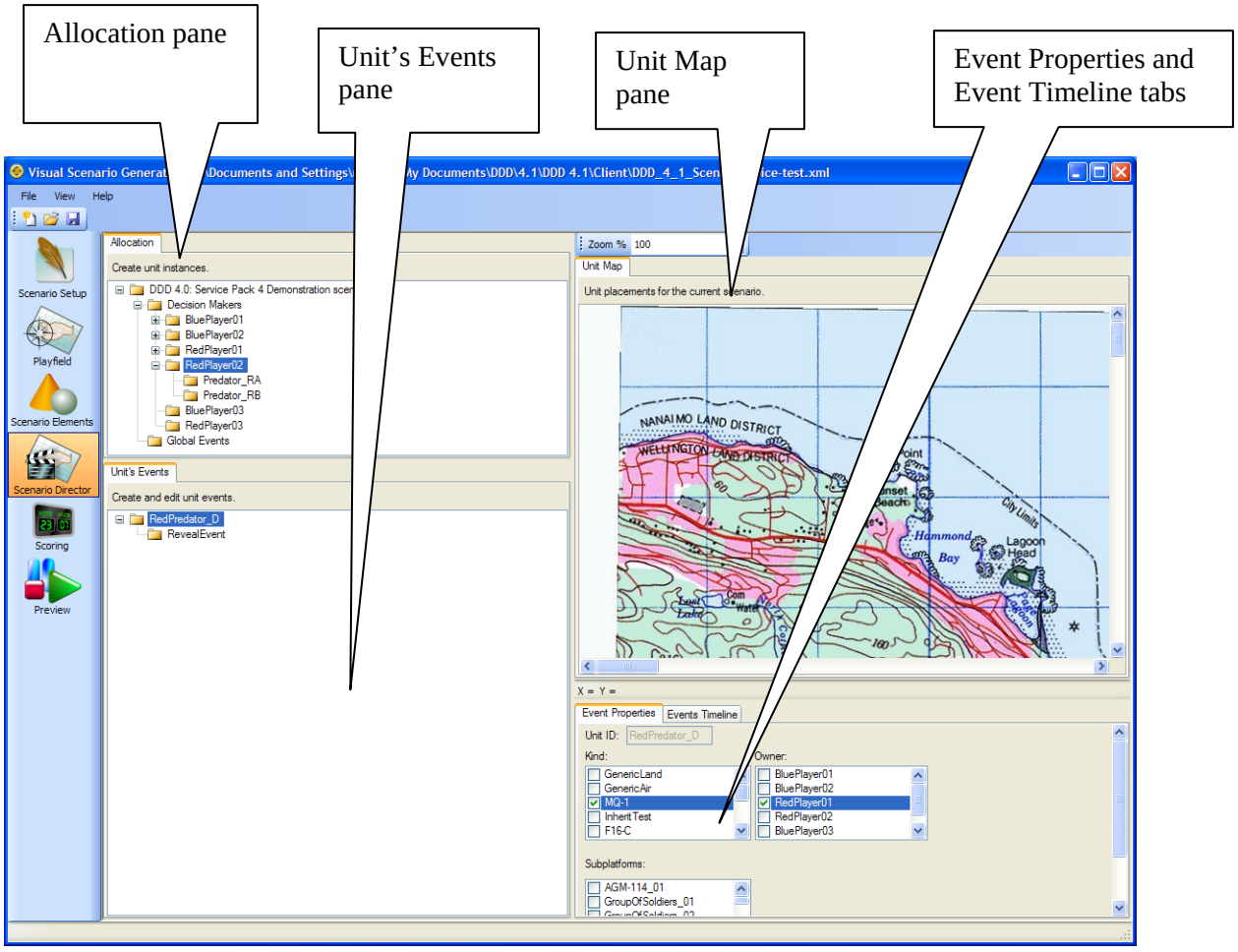
6.0 Putting the Scenario Pieces Together

When using VSG to create a scenario, the first steps are to establish the playfield and the icons to be displayed on it, establish regions in the playfield and specify properties for those regions, and define some scenario elements (such as players, teams, and species definitions). The next step is to put these pieces together by using the following views in VSG:

- Scenario Director view
- Scoring view
- Preview view

6.1 Scenario Director View

In the Scenario Director view, create units (using the scenario elements that the scenario writer has defined), allocate those units to Decision Makers, create global events, create events for individual units, and assign properties to those events.



Use the Scenario Director view to perform the following major tasks:

- Create units and allocate them to players
- Create events for each unit

6.1.1 Creating Units and Allocating Them to Players

Units are the resources represented by icons on the playfield of a scenario. Units are allocated to and controlled by players. The scenario writer can create autonomous objects that are controlled by the scenario. For example, a tank can be defined that is not controlled by a player. This tank can act in its own self-defense and attack objects that belong to teams it considers hostile. Note: even if the scenario writer scripts the events for a unit and allocates that unit to an active DDD player, that player will be able to override the scripting when the DDD is running.

A unit is derived from a Species Definition that the scenario writer created in the Scenario Elements view. One or more units can be created as the type of a single Species Definition.

Use the following steps to create a unit and allocate it to a player:

6. Right click on the name of the Decision Maker to which the unit will be allocated. (The names of all existing Decision Makers are displayed under "Decision Makers" in the tree structure in the Scenario Units pane.) The Create Unit drop-down menu item is displayed.
7. Click Create Unit. The Create Unit dialog box is displayed.
8. Enter a name for the unit in the Name box.
9. Click **Add**. The name of the new unit is now displayed under the name of the Decision Maker to which it belongs in the tree structure in the Scenario Units pane. After a unit is allocated to the Decision Maker, the scenario writer fills in the event properties window which type of unit was added and its specific parameters.

6.1.2 Creating Events for Each Unit

Once a unit is defined, the next step is to define some basic unit properties and then create events for that unit.

Each unit must be specified as being a certain Kind (i.e., type) and has an Owner. The Owner is defaulted to the owner for which the unit was created, but Owner can be changed here (the corresponding Unit Instances tree will update accordingly. The unit may also optionally have Subplatforms.

Subplatforms are units carried by another unit. For example, suppose an aircraft carrier carries planes. The aircraft carrier is a unit, and the planes it carries are subplatforms of the aircraft carrier. Subplatforms are allocated to the same Decision Maker as the unit that carries them. When subplatforms are located on the unit to which they belong, they are *docked*. Some subplatforms, such as planes or weapons, can be launched. Even when a subplatform is launched or *undocked*, it is still considered to be a subplatform belonging to a specific unit. When a subplatform is docked, it can be destroyed only if the unit on which it is docked is destroyed. When a subplatform is undocked, this requirement is no longer true – the subplatform can be destroyed independently.

A weapon is a limited type of subplatform. It is not possible to begin a simulation with any undocked weapons. When weapons are launched, they are directed at a specific target. Once a weapon has been launched, its path cannot be changed. Also, subplatforms can carry weapons. For example, a plane can have a subplatform (a tank) that has its own subplatform (a weapon). Once a subplatform has launched, it is possible for a different Decision Maker to take ownership of it.

There are two broad categories of events:

- Events for a specific unit
- Global events

Global events are not related to individual units. An example of a global unit is the creation of a chat room. (To learn more about chat rooms, see Example 8: Using Global Events, in Section 7.0)

On the other hand, events for a specific unit will cause the unit to be displayed during the simulation and give it the ability to take action. Once events have been defined, their properties can be edited. Events can be used to perform and complete a task.

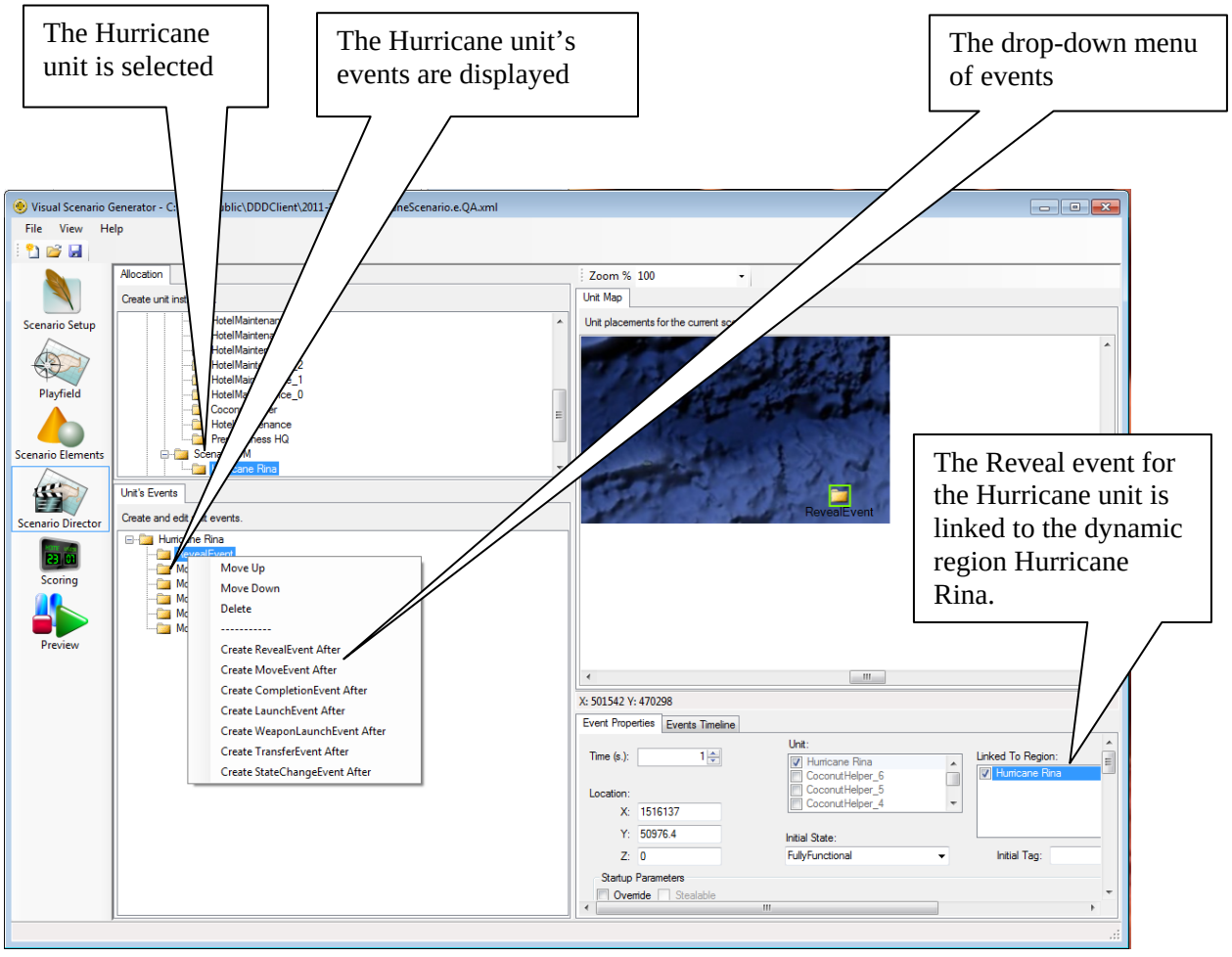
The following are the major types of events that a scenario writer can assign to a unit:

- Reveal Event brings the unit into play at a specific time and location. Every Unit that will appear on the playfield must have a Reveal Event. Reveal Events have times associated with them specifying when the unit will come into existence during the simulation.
- Move Event moves the unit from one location to another.
- Completion Event is an event that happens at the end of a prior event (such as a Move). Other events can be specified to be part of the completion event.

These unit events can be linked to a dynamic region, enabling the region to exhibit "unit like" behaviors during scenario execution that mimic real world, region-based events such as moving weather patterns, wild fires, etc.

To create events for a unit, use the following steps:

1. Click on the name of a unit in the Allocation pane. That unit name is now displayed in the Unit's Events pane.
2. Right click on that unit name in the Unit's Events pane. A drop-down menu of events is displayed, as shown below.



3. To add a Reveal Event for the unit, click Create RevealEvent Child. A RevealEvent node is displayed under the unit name in the tree structure in the Unit's Events pane. At the same time, a RevealEvent icon is displayed in the Unit Map pane. The Reveal Event properties are displayed in the Event Properties tab where they can be edited. These properties include the following:
   - A Unit field – select the unit to be revealed (defaults to currently selected unit).
   - The Time at which the unit is brought into play during the simulation – the time is the number of seconds after the scenario begins that the unit is brought into play. When a Reveal Event is based on a conditional event, the time is the number of seconds after the conditional event ends that the unit is brought into play.
   - The Location, in three dimensions, at which the unit is brought into play during the simulation – the location can be entered as X, Y, Z coordinates in the Event Properties tab. Alternatively, drag and drop the RevealEvent icon in the Unit Map pane.
   - The unit's Initial State, which is Fully Functional by default – the scenario writer can choose a different state.
   - The unit's Initial Tag, which is blank by default. Text entry is optional.
   - Startup parameters
   - The Engram Range – all of the conditions specified in an engram must be satisfied before this unit will be revealed. When an engram range is used, the event is known as a *conditional event*.
4. Right click the RevealEvent node to display the drop-down menu of events again.
5. To add a Move Event, choose Create MoveEvent Child. A MoveEvent node is displayed in the tree, and a MoveEvent icon is displayed in the Unit Map pane. The Move Event properties are displayed in the Event Properties tab where they can be edited. These properties include the following:
   - A Unit field – select the unit to be moved (defaults to currently selected unit).
   - The Time at which the unit should move – the time is the number of seconds after the unit is revealed.
   - The unit's initial Throttle speed – this is a percentage of the unit's maximum velocity. The default is 1, which indicates 100% velocity. For example, to reduce the velocity to 50%, change the value of this property to 0.5.
   - The unit's Destination – the destination can be entered as coordinates in the Event Properties tab. Alternatively, drag and drop the MoveEvent icon in the Unit Map pane.
   - The Engram Range – all of the conditions specified in an engram must be satisfied before this unit will be moved.
6. To add a Completion Event, choose Create CompletionEvent Child. A CompletionEvent node is displayed in the tree, and a CompletionEvent icon is displayed in the Unit Map pane. The Completion Event properties are displayed in the Event Properties tab where they can be edited. These properties include the following:
   - A Unit field – select the unit to be affected by the Completion Event (defaults to currently selected unit).
   - The Action that triggers a new event – when this action occurs, a new event is triggered.
   - The State that triggers a new event – when the unit enters this state, a new event is triggered.

- The Engram Range – all of the conditions specified in an engram must be satisfied before this event will be triggered.

7. To link the unit's event to a dynamic region, select one of the dynamic regions listed in the "Linked To Region" list (located to the right of the Unit list in the Event Properties tab).

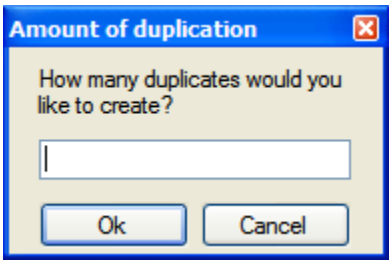
Once events have been created, it is possible to use the Events Timeline pane for information about events and for navigation. See the examples in Section 7.0, Creating and Using Events, for more details about event handling.

6.1.3 Creating Duplicate Units

Units and the events associated with them may be duplicated.

Use the following steps to create duplicates of a unit:

1. Right click on the name of a Unit that will be duplicated. A context menu will appear, listing all of the valid operations that can be performed on the Unit.
2. Select "Duplicate" from the context menu. The Amount of duplication dialog box is displayed.



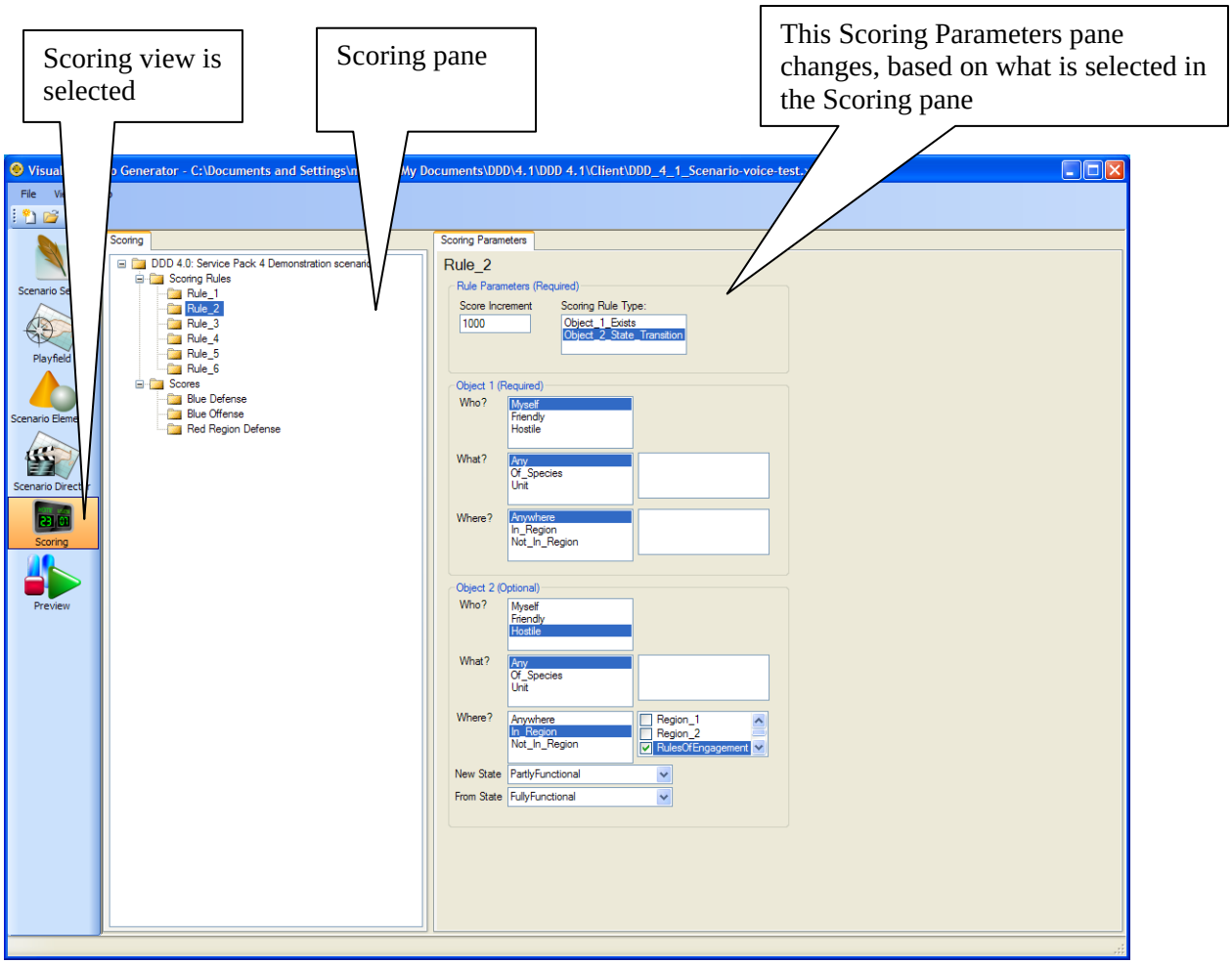
3. Enter the number of duplicates desired. A positive integer.
4. Click **Ok**. The desired number of Units will be created.

A unique name for each duplicated unit will be generated automatically by the VSG. The auto-generated names will take the form **<*name of unit*>_0** through **<name of unit>_(# of duplicates -1).** For example, If 4 duplicates are desired for a unit named ***Fighter***, the VSG will produce the following duplicates: **Fighter_0**, **Fighter_1**, **Fighter_2** and **Fighter_3**

Important: *Unit duplication can be a time consuming process. Completion time for this operation varies with the number of duplicates desired and the number of events linked with the unit.*

6.2 Scoring View

In the Scoring view, create scoring rules and set parameters for scoring. Because a simulation is like a game, its players often have goals that can earn points during the simulation. The players can be given a final score when the simulation ends. The scenario writer is responsible for setting up the scoring system for each scenario.



There are two tasks to complete when setting up a scoring system:

- Create scoring rules.
- Use the scoring rules to determine how a score will be composed for each player.

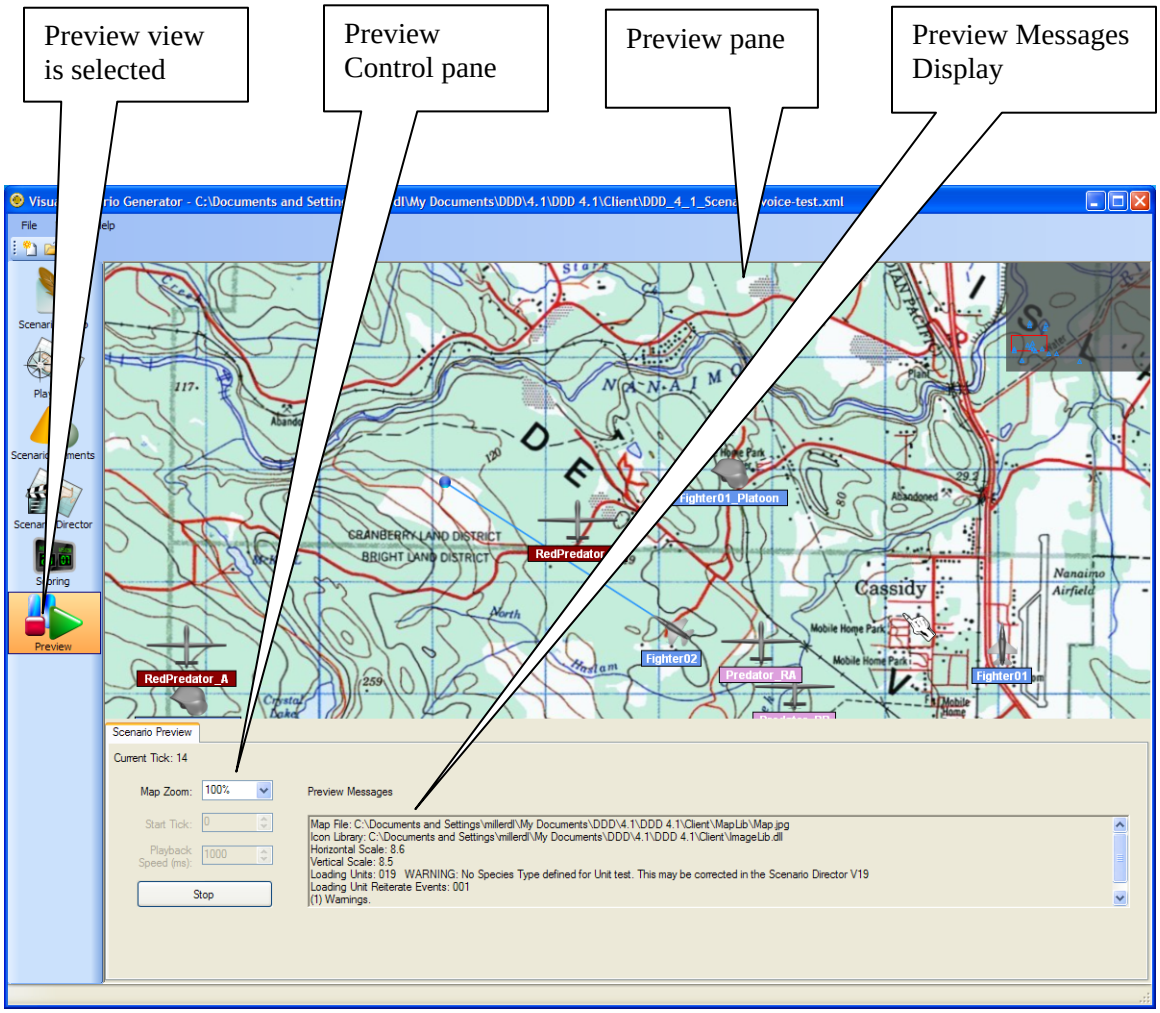
Inline Help provides step-by-step instructions for creating scoring rules and using them. It is important to understand that there are two types of rules: the Existence rule and the State Transition rule, both of which are described in detail in Inline Help. Also, note that when editing scoring rules and scores, some parameters are required and others are optional.

Click Scoring Rules in the Scoring pane to display Inline Help for scoring rules, including detailed examples. Click Scores in the Scoring pane to display Inline Help that explains how to compose a score for each player.

6.3 Preview View

In the Preview view, use VSG to preview the scenario to find out what it will look like when players play a simulation. The Preview view shows the playfield, the icons, and scripted actions.

Upon entry into Preview view, Preview view will begin initializing the scenario for playback. The initialization process can be time consuming process, depending on the complexity of the scenario. Refer to the Preview Messages display to monitor the initialization progress.



Use the following steps to preview the scenario in its current state:

1. In the Preview Control pane, set the following options:
   - Map Zoom changes the percentage of the scenario map that is displayed.
   - Timetick (in milliseconds) changes the speed at which the scenario runs – this specifies the number of milliseconds between each time tick. The lower this number (say, 250) the faster the preview moves (1 tick every 250 milliseconds).
   - Start Tick specifies at what time in the scenario to begin (for example, the value 0 will play the scenario from its beginning)
2. Click **Play**. The scenario will play in the Preview pane.

Previewing the scenario in VSG means checking to see how the scenario works before it is used by players in a simulation. Previewing gives the scenario writer the opportunity to check for

errors in the scenario and to make sure that the units appear and behave as expected. After using Preview, scenario writers can continue working on the scenario in an iterative way, for example, by defining more scenario elements. Once the scenario is complete and the preview is satisfactory, it is a good practice to test the scenario using a DDD simulation. The preview mode is limited to pre-scripted scenario events. When the DDD runs the scenario, all of the user interactions and simulation conditions can be tested.

7.0 Advanced Topics

Once a scenario writer understands the basic process of creating a scenario with VSG, there are a few advanced topics that are important to understand.

7.1 Units and Inheritance Rules

Remember that every unit is based on a Species Definition. There are specific inheritance rules that are important to know when creating units. Knowing the inheritance rules allows a scenario writer to predict and use the full power of how a unit inherits (or does not inherit) states from its Species Definition.

A unit's identity is tied closely to its states. This is what makes it possible for the unit's abilities to change when a simulation is played. A unit has two states by default: Fully Functional and Dead. In some cases, these may be the only states needed. But when a scenario writer defines new states for a unit, that unit gains new ways to behave. For example, a police squadron might have participated in a training exercise on multi-cultural awareness that has given it a new state of "Diverse Urban Capable" with new negotiation capabilities. It is possible to define a state that causes a unit to lose some of its abilities. Imagine that after an engagement, a tank might be in a "Damaged" state. Another state can be defined so that unit can regain some or all of its lost abilities. It is even possible to define a state that lets a unit gain more abilities that it has when it is in the Fully Functional state.

NOTE: It should be noted that all user-created states will inherit from the FullyFunctional state. This gets very confusing when the FullyFunctional state is used as a PLAYABLE state, rather than a TEMPLATE state which defines attributes true for ALL states. Aptima suggests using the FullyFunctional state only as a template to provide basic attributes true for all states.

In order to define a variety of new states, the scenario writer must understand the ways in which a unit instance inherits attributes from the Species Definition. There are two types of inheritance:

- State Inheritance (inheritance related to the hierarchy of the Species Definition)
- Attribute Inheritance (inheritance from a unit's Fully Functional state to other states)

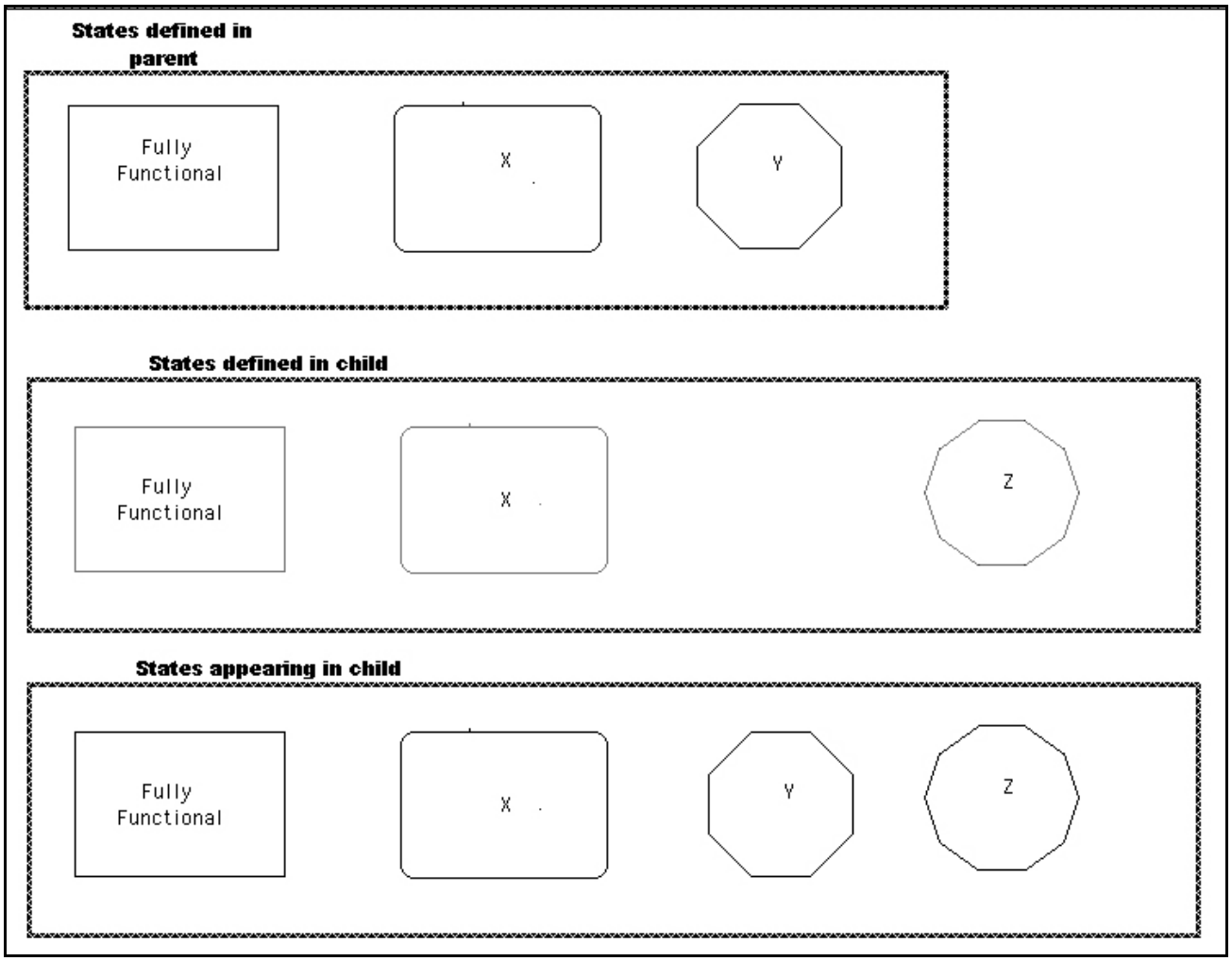
State inheritance happens when one Species Definition is based on another Species Definition. In this case, when the new unit type is defined, it has the same states as the Species Definition used to define that original base unit. For example, it is not necessary to define the Fully Functional and Dead states for units – these are default states, so the unit will inherit them.

Attribute inheritance begins with Fully Functional state of the Species Definition used to define a unit. *Any definitions made for the new unit will override the definitions inherited from the Species Definition.* It is also important to understand that some attributes are inherited differently from others. The attribute values in the Fully Functional state are overridden by any values in the unit, on a one-by-one basis, with the following exceptions:

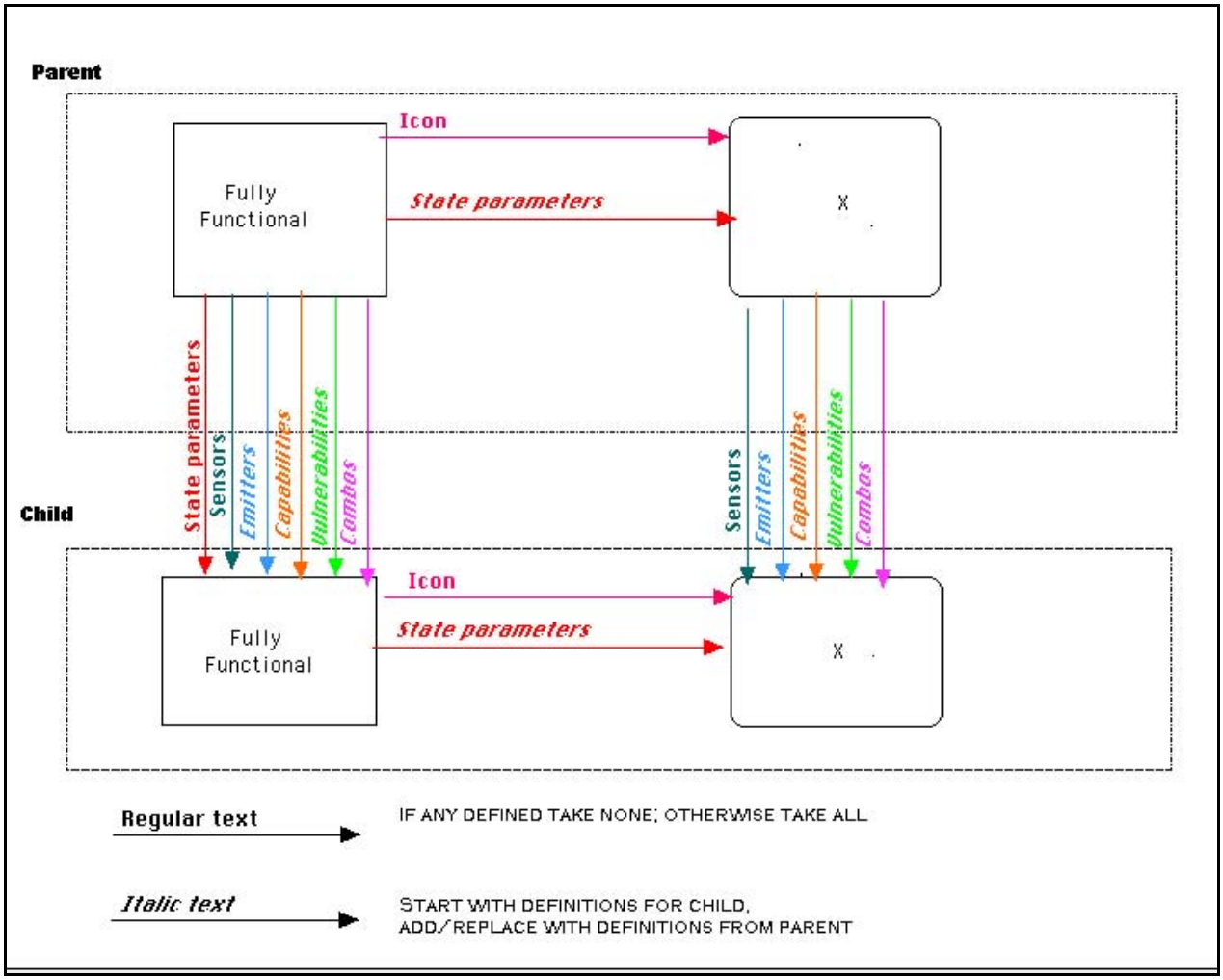
- If a unit has no sensors defined, then it inherits *all* the sensors of the Species Definition.
- If a Unit has *any* sensors defined, it inherits *none* of the sensors from the Species Definition.

- The other states of the unit (except the Dead state) inherit any values from the Fully Functional state that are not explicitly defined for it.

The following illustration shows the general process of inheritance, based on which states are defined for a type of unit (the child) and which states are defined in the Species Definition (the parent).



The following illustration shows the detailed rules of inheritance.



7.2 How Capabilities Define States

A unit's capabilities give it the ability to change another unit's state. For example, suppose a scenario has three tank units called Tank1, Tank2, and Tank3. The scenario writer can write a scenario in which Tank1 destroys Tank2 (meaning, Tank1 pushes Tank2 into the Dead state). At the same time, if Tank1 takes the same action against Tank3, the result is minimal or no damage to Tank3.

Remember that a unit can have many capabilities, and each capability is named. Each capability has a collection of distance-related effects and can combine effects with other units. For example, a scenario writer can specify the following restrictions for the Destroy capability:

- This capability can be used only when one unit is within two kilometers of another unit.
- If the player chooses to use the Destroy capability, there is a 70% chance that it will be successful in destroying the targeted unit.
- The intensity of its effect is 100. This intensity value is crucial to making capabilities and vulnerabilities work together.

The intensity value represents an amount of "force" that is applied to another unit. For some units, an intensity value is high enough to push them into the Dead state. For other units, the same

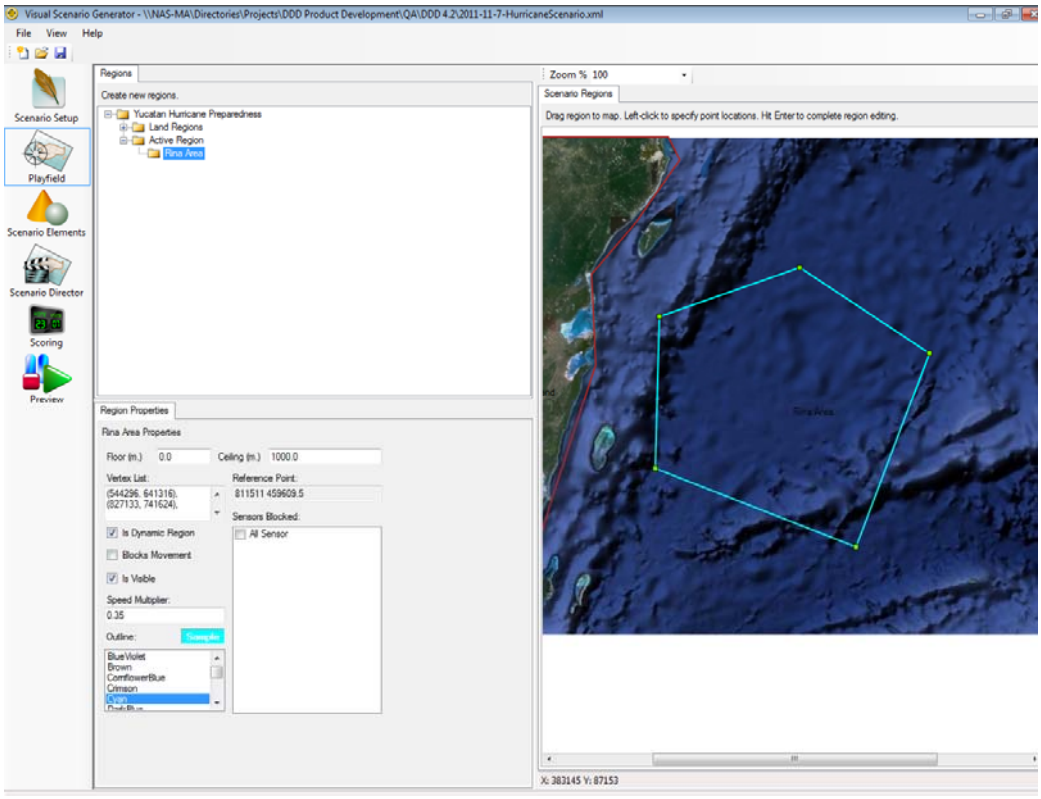
intensity might only disable them, and the same value might have no effect at all on a third group. The impact on each unit is determined by the vulnerabilities defined for that unit.

8.0 Examples: Creating and Using Events

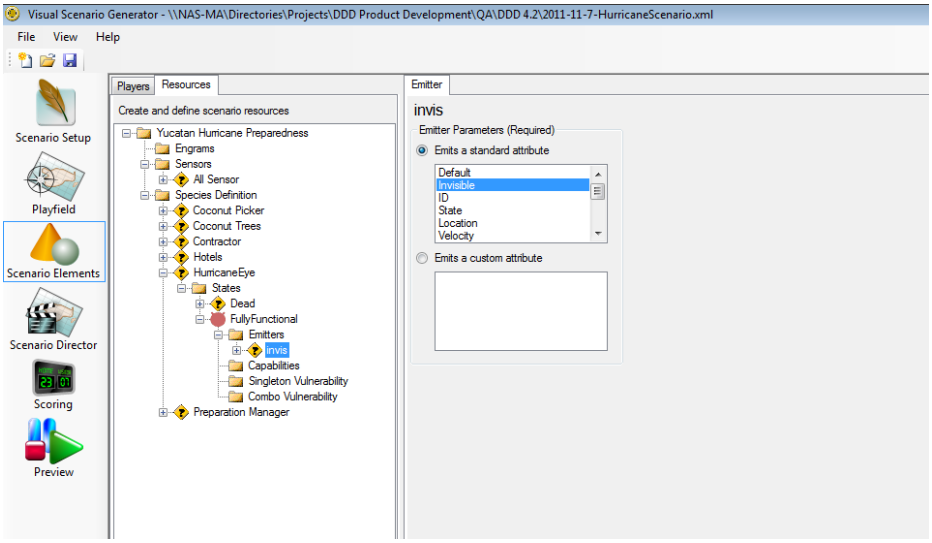
A general description of defining events and their uses is in the section Creating Events for Each Unit on page 23. This section gives specific examples of some common types of events that scenario writers need to create.

Read the examples in this section, as well as the examples in The use of dynamic active regions enables the scenario writer to mimic changing environmental patterns (e.g., weather, wildfire) and create corresponding obstructions and sensor blocks. Defining a dynamic active region is a multi-part process, as described in the hurricane example below.

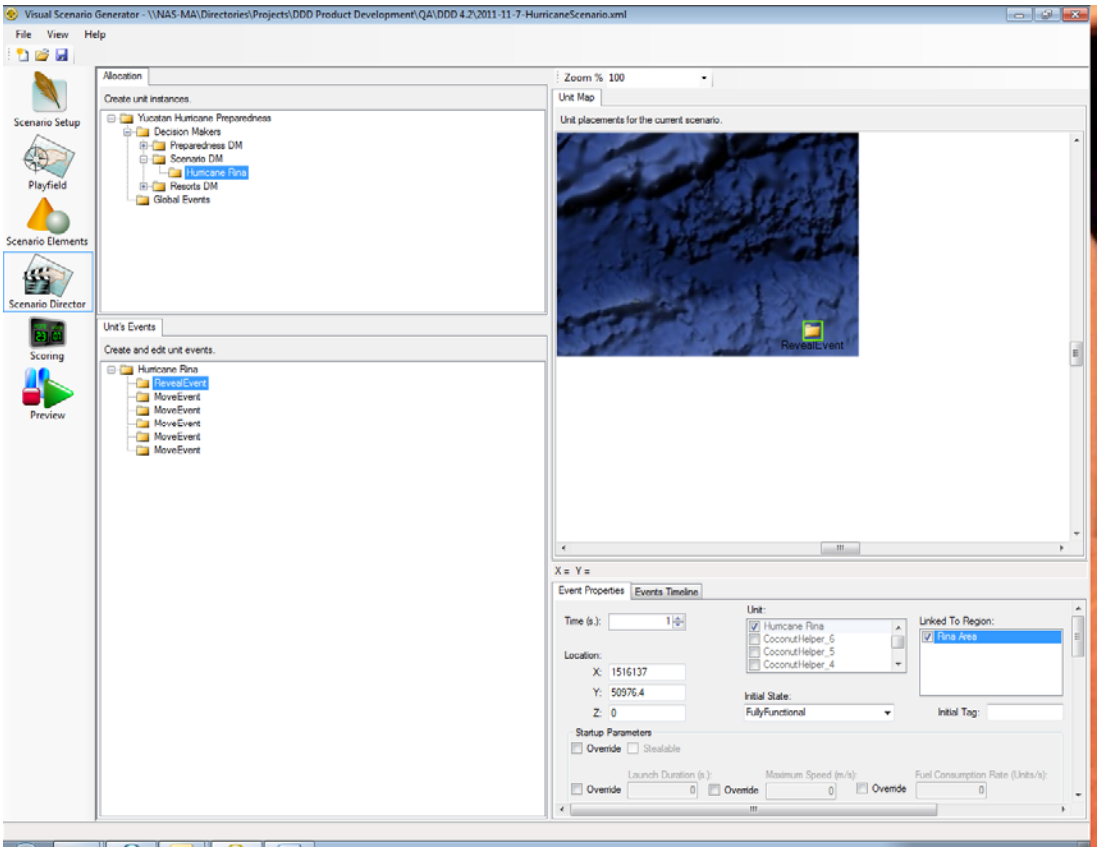
In the Playfield view, the scenario writer creates an active region and specifies it as dynamic by clicking the "Is Dynamic Region" checkbox in the Region Properties tab as shown in the figure below.



In the Scenario Elements view, the movement of the region is defined by creating an object capable of movement (e.g., HurricaneEye in this example). Optionally, the object's emitters can be specified as invisible to suppress display of the object in the DDD client views. To the player, the region will appear to move without the underlying object being shown.



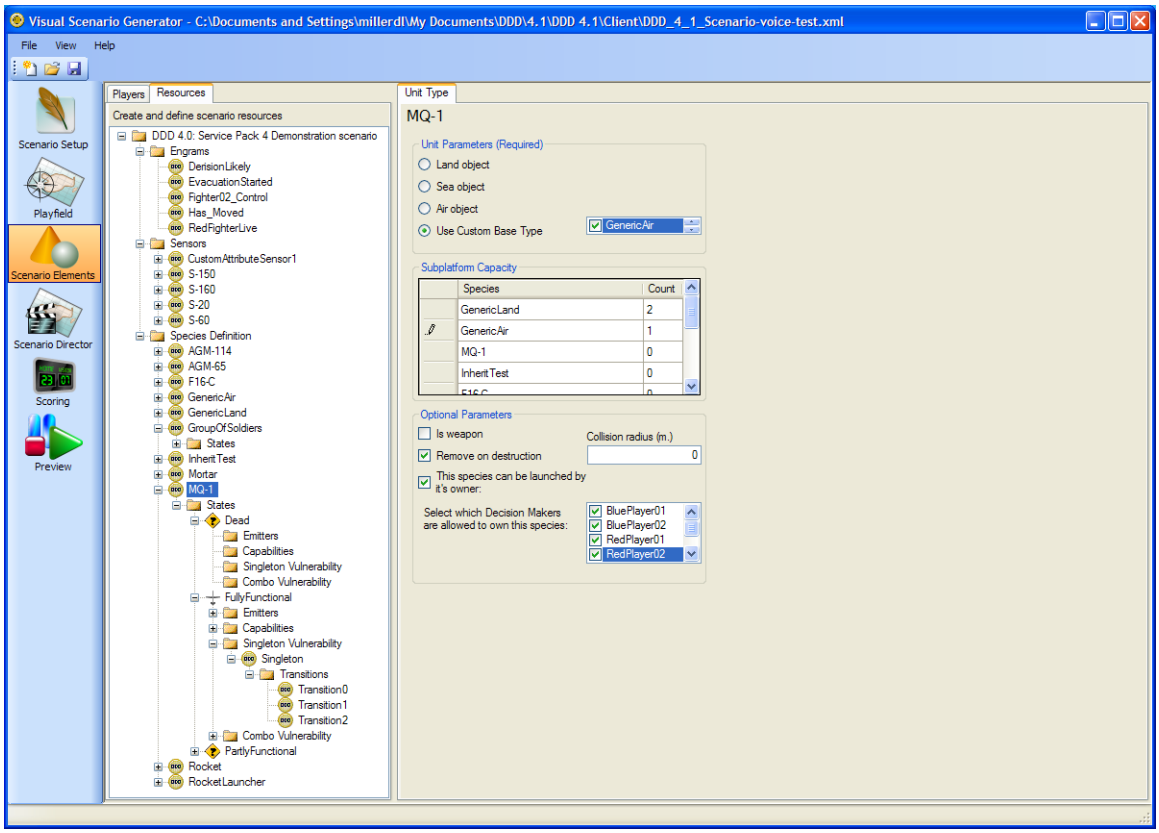
Once both the dynamic active region and the corresponding object have been defined, the object is instantiated in the Scenario Director view (Hurricane Rina in the example below). Then, a series of reveal and move events are defined that specify the behaviors for the linked active dynamic region (Rina Area in the example below).



9.0 Frequently Asked Questions (beginning on page 49) to learn about events and event handling. For more details, see the Scenario Writer's Guide, which is included in the DDD documentation set.

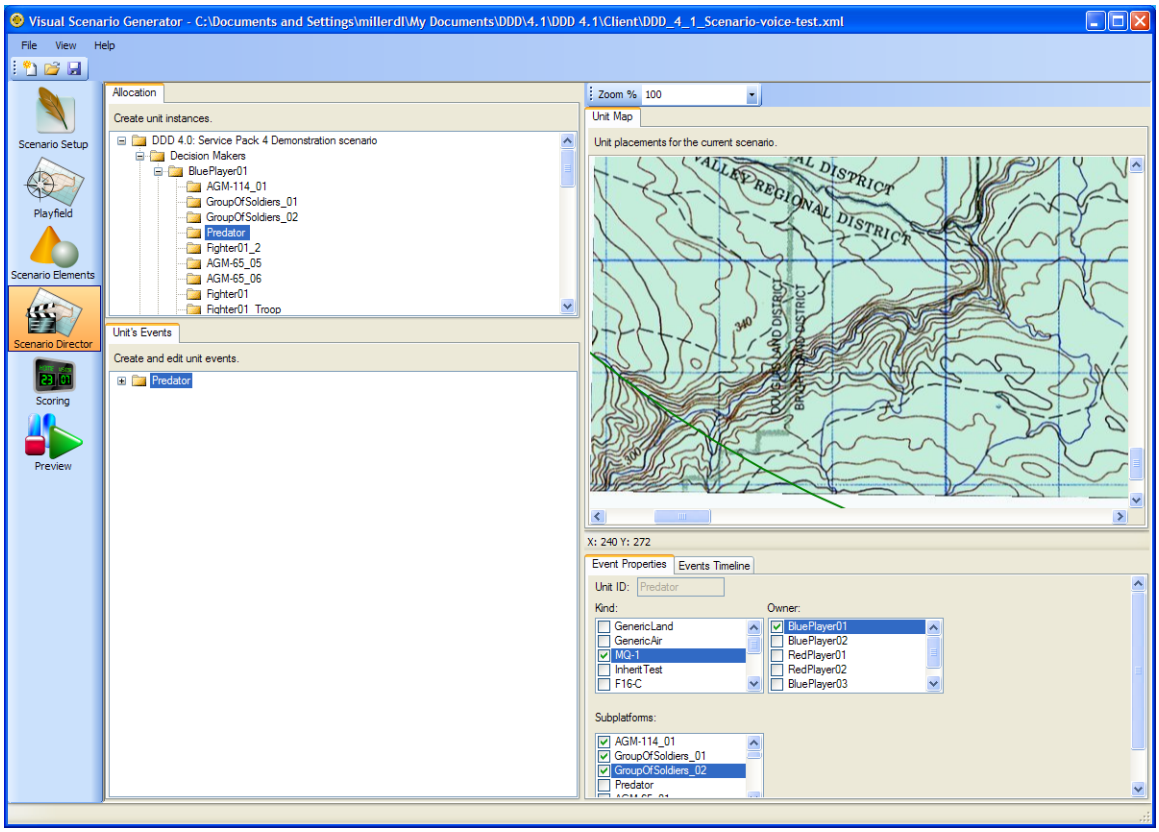
Example 1: Creating Subplatforms

Remember that before events can be defined, a Decision Maker and a Species Definition must already have been defined in the Scenario Elements view. In this example, "BluePlayer01" is the Decision Maker and several species have been defined including "MQ-1", "AGM-114_01", "GroupOfSoldiers_01", and "GroupOfSoldiers_02". MQ-1 is a GenericAir object that can include Subplatforms as specified in the Subplatform Capacity table – in this example, 2 species of type GenericLand and 1 of type GenericAir. Only species for which at least one non-zero Subplatform Capacity has been specified can contain any subplatforms.



Once the set of species to be used in the scenario have been defined, use the tree in the Allocation pane of the Scenario Director view to create units and corresponding unit or global events.

To define a new unit, right click on a Decision Maker (in the tree in the Allocate pane), choose Create Unit from the dropdown menu that is displayed, and assign a name to the new unit. By default, the new unit has no events. In this example, BluePlayer01 includes the unit "Predator". Note that Predator is selected in the Unit's Events pane, which causes its properties to be displayed in the Event Properties pane. The first task is to specify the Kind of species of the unit. In the example below, "MQ-1" has been selected in the Kind box. The unit's Owner (in this case, BluePlayer01) is selected by default.

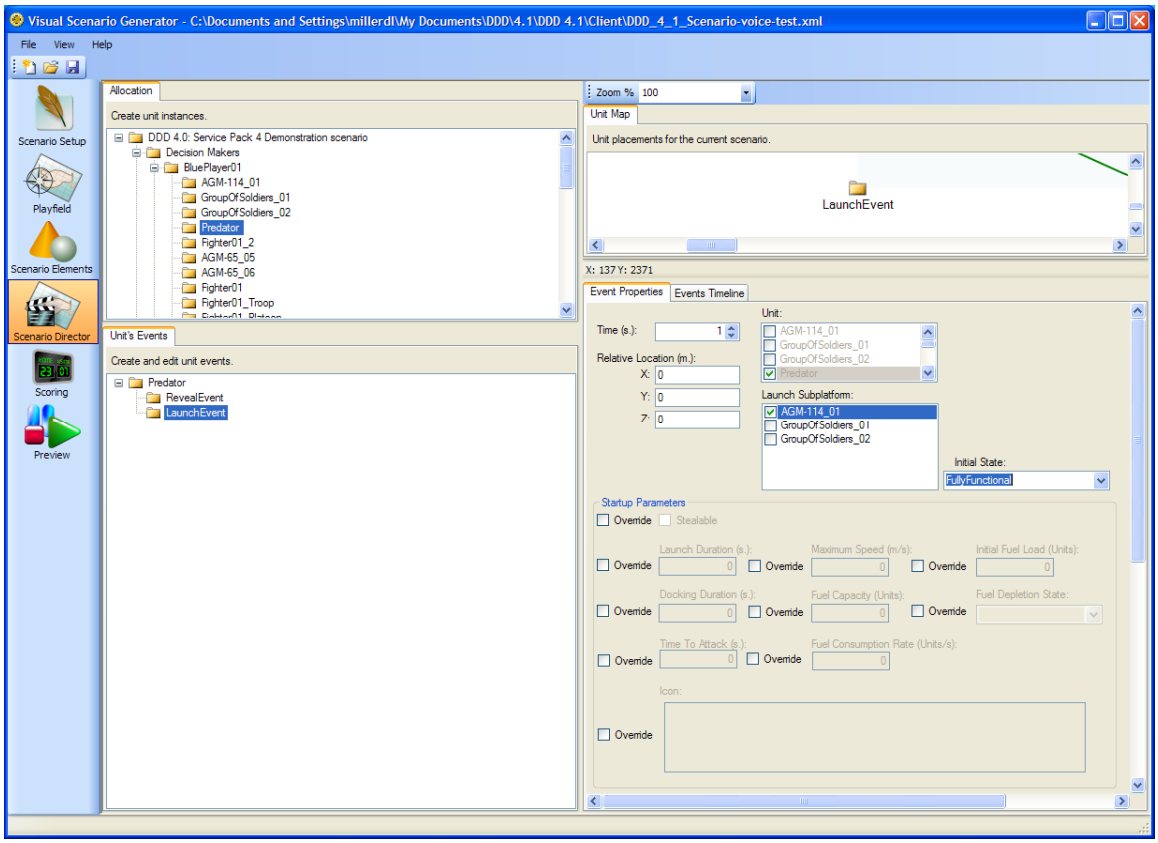


Note: It is possible to change the owner of a unit after that unit has been defined. To do so, select the unit in the Unit's Events pane, then select the new Decision Maker in the Owner box. The new Decision Maker must be defined before it can be displayed in the Owner box. When a unit is reallocated, its node will automatically move under the node of its new owner in the Allocation tree.

Recall that the species definition for an MQ-1 included a Subplatform Capacity of 2 GenericLand objects and 1 GenericAir object. This means that any unit of type MQ-1 can optionally include up to that many subplatforms (of those types). The Subplatforms list in Event Properties enables the scenario designer to specify a set of Subplatforms to associate with the unit being edited. The list consists of all existing (i.e., already created) units in the Allocation tab.

In this example, the MQ-1 unit has several Subplatforms selected including 1 air unit, "AGM-114_01", and 2 land units, "GroupOfSoldiers_01", and "GroupOfSoldiers_02". To include multiple units of the same species as subplatforms, multiple units must be created in the Allocation tab. Thus, in this example, to include two GroupOfSoldiers units as subplatforms, at least two units of that type must exist in the Allocation tab.

When the Predator unit is revealed during a scenario run, its three subplatforms will not be displayed until they are launched – either via a LaunchEvent specified in the scenario itself or in realtime when a Decision Maker initiates a Launch action. In other words, the AGM-114 and two GroupOfSoldiers are connected to the Predator. The AGM-114 and GroupOfSoldiers units must be launched from the Predator before they can become active in the scenario.



To specify a LaunchEvent, select the parent unit in the Allocation tree and then right click on it in the Unit Events tab. Select Create LaunchEvent Child to edit the Launch Event properties in the tab to the right.

When Predator is created, it will be revealed on the screen. As specified above, subplatform AGM-114_01 also will be displayed on the screen at Relative Location specified (i.e., relative to the unit it is being launched from). In this example, the AGM-114_01 is a unit that is pre-launched from the Predator when the scenario begins at Time 1. AGM-114_01 acts like a subplatform that is not docked when the scenario begins, but by editing the Time, the subplatform could start out in a docked state at the onset of the scenario and be automatically launched at the time specified. The Initial State of the subplatform upon its launch, defaulted to FullyFunctional, can be changed to any other state defined for that species.

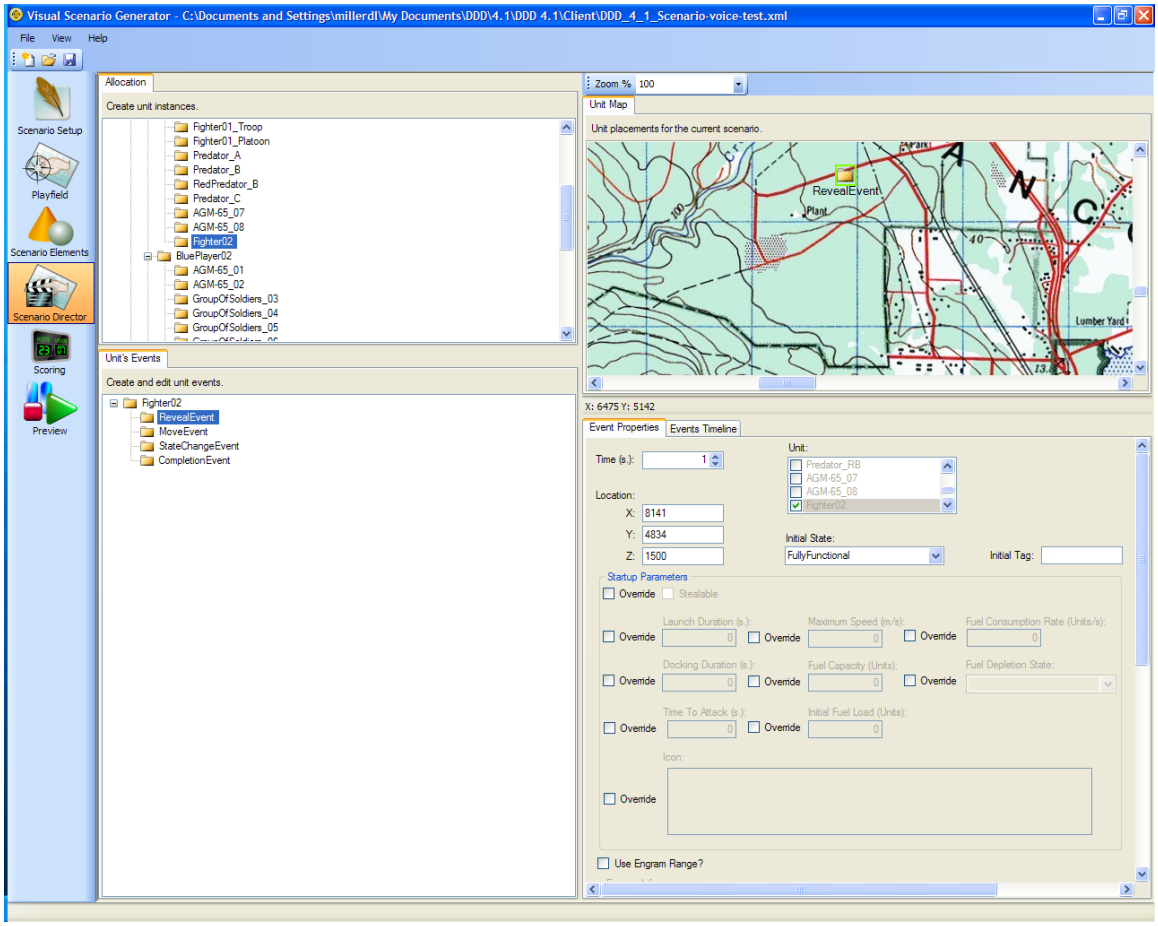
Notice the default startup parameters for AGM-114_01 in the example above, any of which can be changed by selecting Override and entering a new value for the parameter.

Example 2: Creating Events for a Unit Controlled by the Scenario

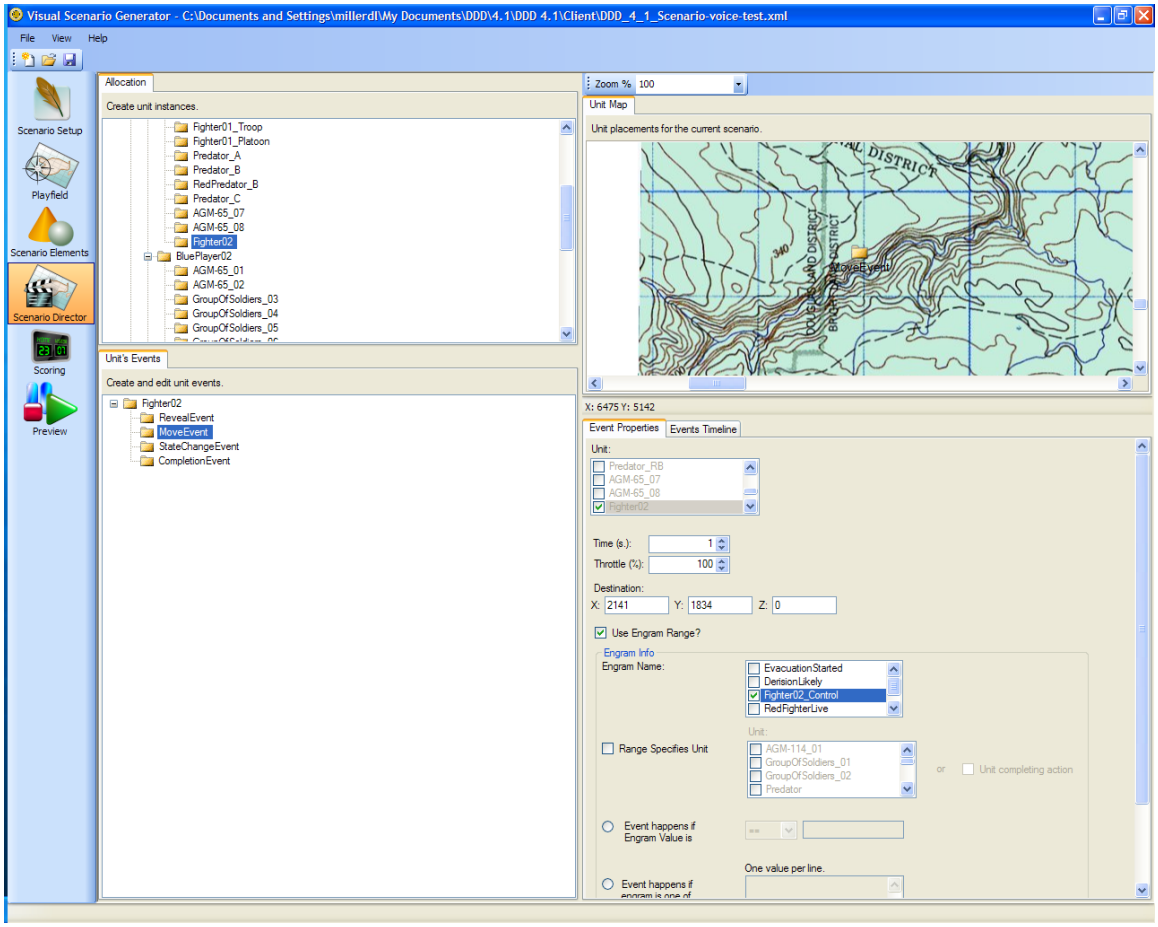
A scenario can be written so that each team has human players who take on the roles of Decision Makers and control the units allocated to those Decision Makers. However, a scenario also be written so that one team has human players and the other team's units are controlled by the scenario itself. This example shows how to create events for a unit named Fighter02, which is

controlled by the scenario. The goal is to have Fighter02 appear on the map at a specific time and then move in a specific pattern.

Select the Fighter02 unit in the Allocation tree. To create a reveal event, right click on Fighter02 in the Unit's Events pane and choose Create RevealEvent Child from the dropdown menu. In this example, the Reveal Event for Fighter02 uses the default Time of 1. This means that Fighter02 will be displayed on the map one second after the simulation begins. Click on the RevealEvent icon (displayed in the lower left corner below the Unit Map) and drag it to the location where Fighter01_Troop_01 should be revealed. Select its initial state as Fully Functional. None of its state parameters should be overridden.



Now that Fighter02 has a Reveal Event, the next step is to create another event that will move Fighter02 from the location where it is revealed to a new location. To do so, right click on the RevealEvent node under Fighter02in the Unit's Events pane, and then choose Create MoveEvent After from the dropdown menu. A MoveEvent node is now displayed under RevealEvent in the Unit's Events pane. A MoveEvent icon is displayed below the Unit Map. Drag and drop the MoveEvent to the desired location – this is the point to which Fighter02 will fly after being revealed. Click on the MoveEvent node in the tree to display its properties.



Note: Any unit can be revealed and moved at the same time. In this example, one second after the simulation begins, Fighter02 will simultaneously be displayed on the map and begin to move to its next location. It is important to understand that Time 0 is reserved as the initialization time of the simulation, and Time 1 is the start time for events. If a unit has move events that have a Time value that is lower than the Time value for the reveal event, those move events will be ignored.

Now that Fighter02 has been revealed and is can move toward its first location, additional move locations can be specified by creating other Move Events. Right click on the MoveEvent node in the tree, choose Create MoveEventAfter from the dropdown menu. Click on the MoveEvent node in the tree to display its properties, and specify 10 as the Time value for this new Move Event. Click on the new MoveEvent icon in the Unit Map and drag it to the desired location.

Now that two Move Events have been created for Fighter02, this example will show a different way to move Fighter02. There are two ways to script Move Events:

- Use absolute time values – this means specifying the Time value in the Event Properties pane for the Move Event
- Use completion logic – this means creating a Completion Event

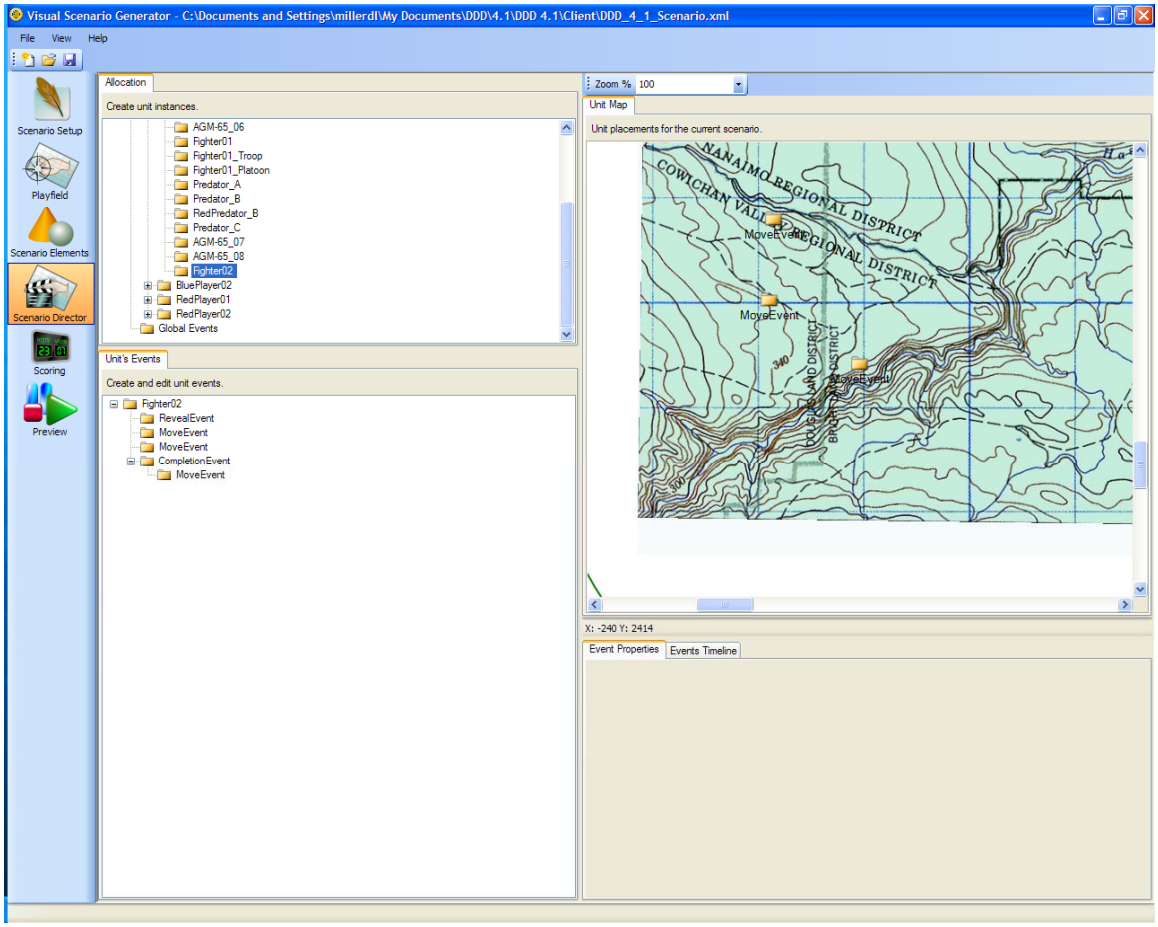
Every time Fighter02 moves to a different point on the map, a new Move Event must be created to specify that point. When using absolute time values, set the Time value for that Move Event. In this example, Fighter02 can begin moving to the first point at Time 1, and then move to the second point at Time 10.

On the other hand, it is possible to use a Completion Event to determine when Fighter02 will move from one point to another. By using a Completion Event, the scenario writer can specify that Fighter02 will begin moving toward its next location after it arrives at its current destination.

To create a Completion Event in this example, right click on the last MoveEvent node in the Unit's Events pane and choose Create CompletionEvent After from the dropdown menu. A CompletionEvent node is displayed in the Unit's Events tree and click on CompletionEvent to display its properties in the Events Properties pane.

In the Events Properties pane, Fighter02 is selected in the Unit field. In the Condition field, select the Action radio button and choose MoveComplete_Event from the dropdown menu.

Right click on CompletionEvent in the Unit's Events pane and choose Create MoveEvent Child from the dropdown menu. A MoveEvent node is displayed as a child of CompletionEvent in the tree (as shown in the illustration below), and a MoveEvent icon is displayed below the Unit Map. Click on the MoveEvent icon and drag it to the desired location on the map. The Completion Event specifies that when Fighter02 arrives at the previous location (meaning, the MoveEvent above CompletionEvent in the Unit's Events tree), it will begin to move toward the next location (the MoveEvent below the CompletionEvent in the tree).



In this example, Fighter02 has been scripted to perform the following actions:

1. At Time 1, Fighter02 is displayed on the map (by creating a Reveal Event with a Time 1 value).
2. At Time 1, Fighter02 begins moving toward its first destination (by creating a Move Event with a Time 1 value).
3. At Time 10, Fighter02 begins moving toward its second destination (by creating a Move Event with a Time 10 value). Note that Fighter02 begins moving toward its second destination regardless of whether it has or has not arrived at its first destination.
4. Once Fighter02 arrives at its second destination, it begins moving toward its third destination (by creating a Completion Event that has a Move Event child).

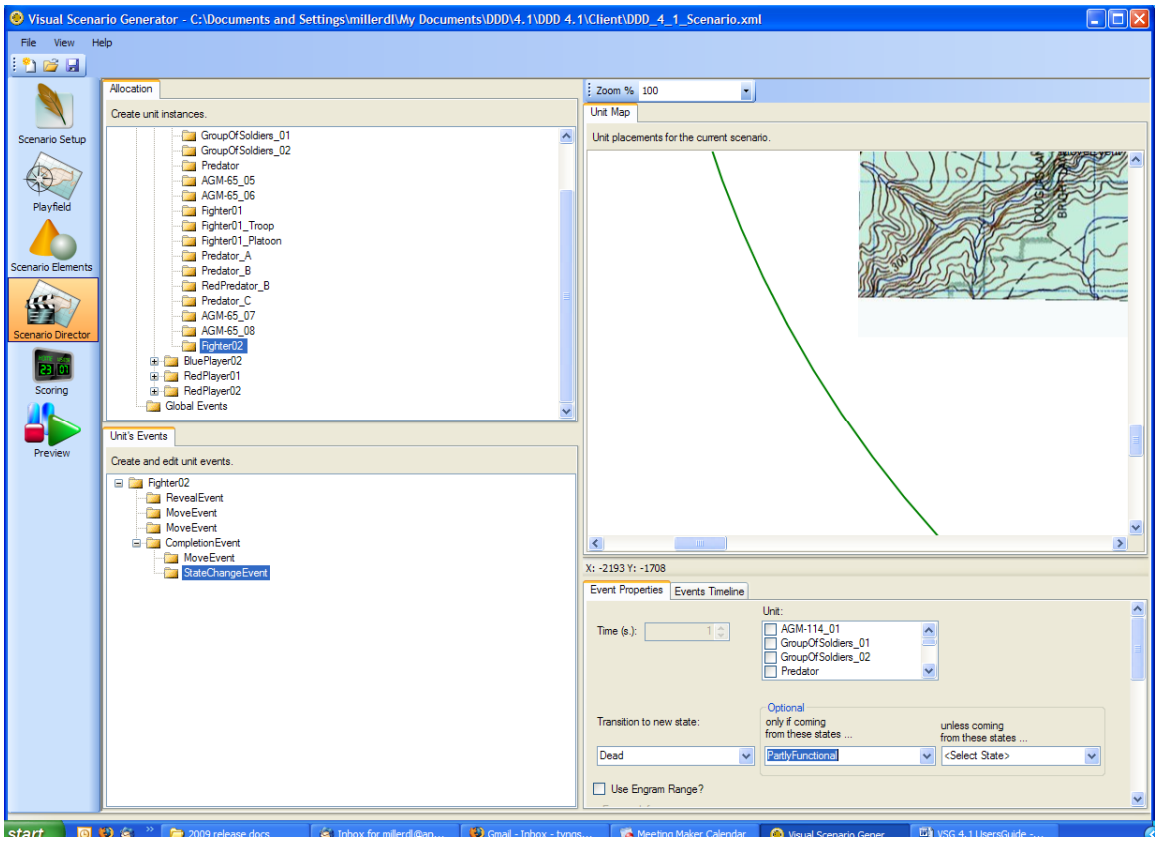
If the scenario writer needs to have specific events happen at specific times, then it is a good practice to use absolute times. On the other hand, to achieve smooth movement of units, it is better to use Completion Events.

Another advantage of using a Completion Event is that it gives the scenario writer the flexibility to create different events for different units once the previous event is completed. For example, once Fighter02 arrives at its second destination, events can be created not just for Fighter02 but also for any other units owned by the Decision Maker who owns Fighter02. This makes it possible to coordinate the actions of multiple units. To do so, select one unit in the Unit field in the Events Properties pane, choose the Action radio button, and then select an event from the dropdown menu.

Example 3: Changing a Unit's State

By default, a unit begins in a Fully Functional state. Instead of creating another event for a unit in a Completion Event, it is possible to change that unit's state. For example, instead of creating an event that causes a unit named Fighter02 to move from its second location to a third location, it is possible to change Fighter02's state to Dead (or another state). To do so, select the Completion Event in the Unit's Events tree. In the Events Properties pane, select the State radio button in the Condition field and then select the desired state.

Another way to change a unit's state is to create a StateChange Event, which gives the scenario writer the flexibility of making the state change conditional. To create this event, right click on the CompletionEvent in the Unit's Events tree and choose Create StateChangeEvent After from the dropdown menu. To display the properties, select the StateChangeEvent node in the tree. As shown below, the scenario writer can choose options to determine if the unit will change its state. For example, although every unit has two default states (Fully Functional and Dead), it is possible to define other states, such as PartlyFunctional. Therefore, it is possible to create a StateChange Event that specifies that a unit's state will be changed to Dead only if it is already in its PartlyFunctional state – otherwise, the unit will remain Fully Functional.



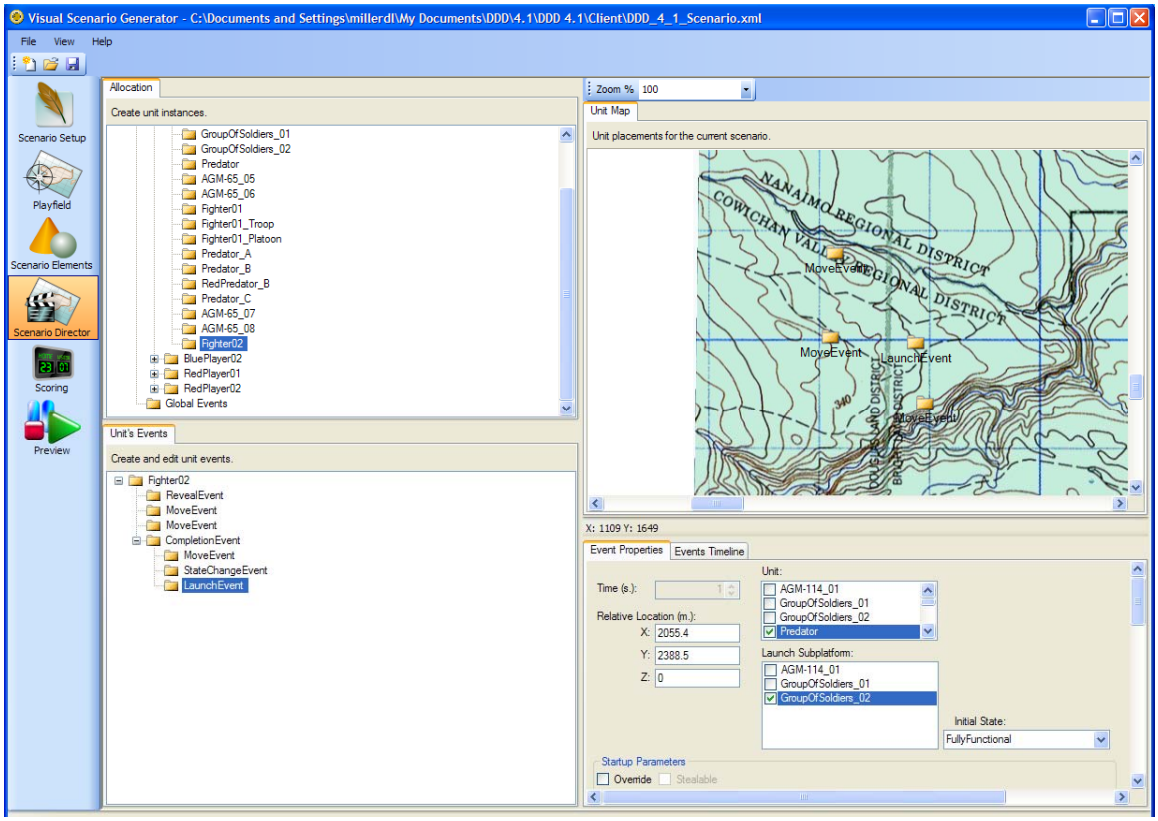
Example 4: Launching Subplatforms

Remember that a subplatform is a unit that is carried by another unit. Subplatforms do not appear on the map – and cannot be used – until they have been launched by the unit on which they reside. This example explains how to launch subplatforms.

As described in Example 1, Predator is a unit that has three subplatforms. When the simulation begins, Predator is displayed on the map but its air and land subplatforms are not. To launch them after Predator is revealed, right click on RevealEvent in the Unit's Events pane and choose Create LaunchEvent After. A separate LaunchEvent is needed for each subplatform to be launched. In this example, create three LaunchEvents if you want to launch all three subplatforms. Repeat the following steps for each subplatform to be launched.

In the Unit field to the right, Predator should be selected by default. To launch a subplatform as soon as the simulation begins, use the default value of Time 1. Select the subplatform to be launched from the Launch Subplatform list.

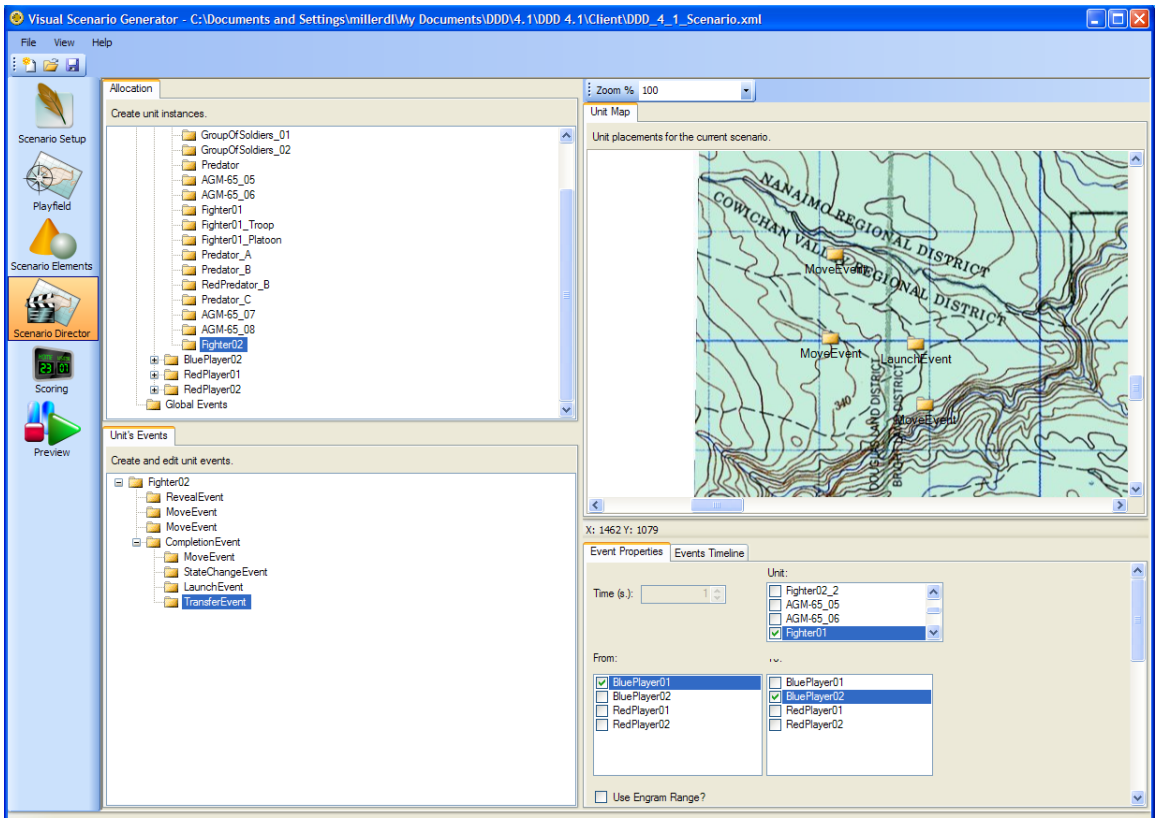
Subplatforms can also be launched as part of a Completion Event. In Example 3, Fighter02 has Move Events and a Completion Event. The Completion Event has a StateChange Event child, which changes Fighter02's state to Dead. In this example, once Fighter02's state changes to Dead, the Predator unit's GroupOfSoldiers_02 subplatform will launch and be displayed on the map and will be able to participate in the scenario. To create a Launch Event, right click on StateChangeEvent and choose Create LaunchEvent After from the dropdown menu. As shown in the illustration below, a LaunchEvent node is displayed in the tree and its properties are displayed.



In the Events Properties pane, select Predator in the Unit field. This specifies that Predator is the unit that will launch a subplatform. The Time value of 1 means that the subplatform will be launched one second after the Launch Event is triggered. In the Launch Subplatform list, select GroupOfSoldiers_02 as the platform to be launched by Predator. Now, when Fighter02 reaches its final destination, its state will change to Dead, and Predator will launch its subplatform GroupOfSoldiers_02.

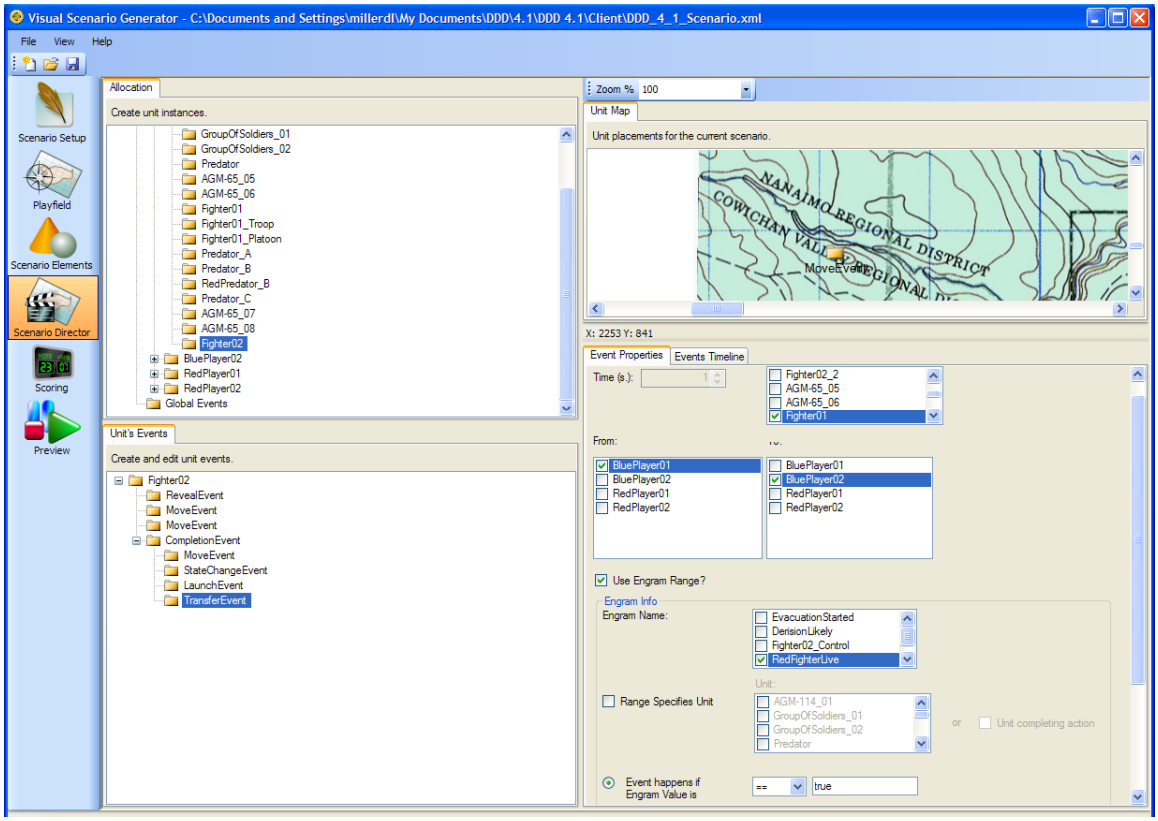
Example 5: Transferring Ownership of a Unit During the Simulation

It is also possible to transfer ownership of a unit from one Decision Maker to another while the simulation is running. For example, after Predator has launched its subplatform, the scenario writer can specify that the ownership of Fighter01 will transfer from BluePlayer01 to BluePlayer02. To do so, right click on LaunchEvent, choose Create TransferEvent After, and specify the transfer of ownership in the Events Properties pane, as shown below. Note: be sure to select the unit to be transferred in the Unit field.

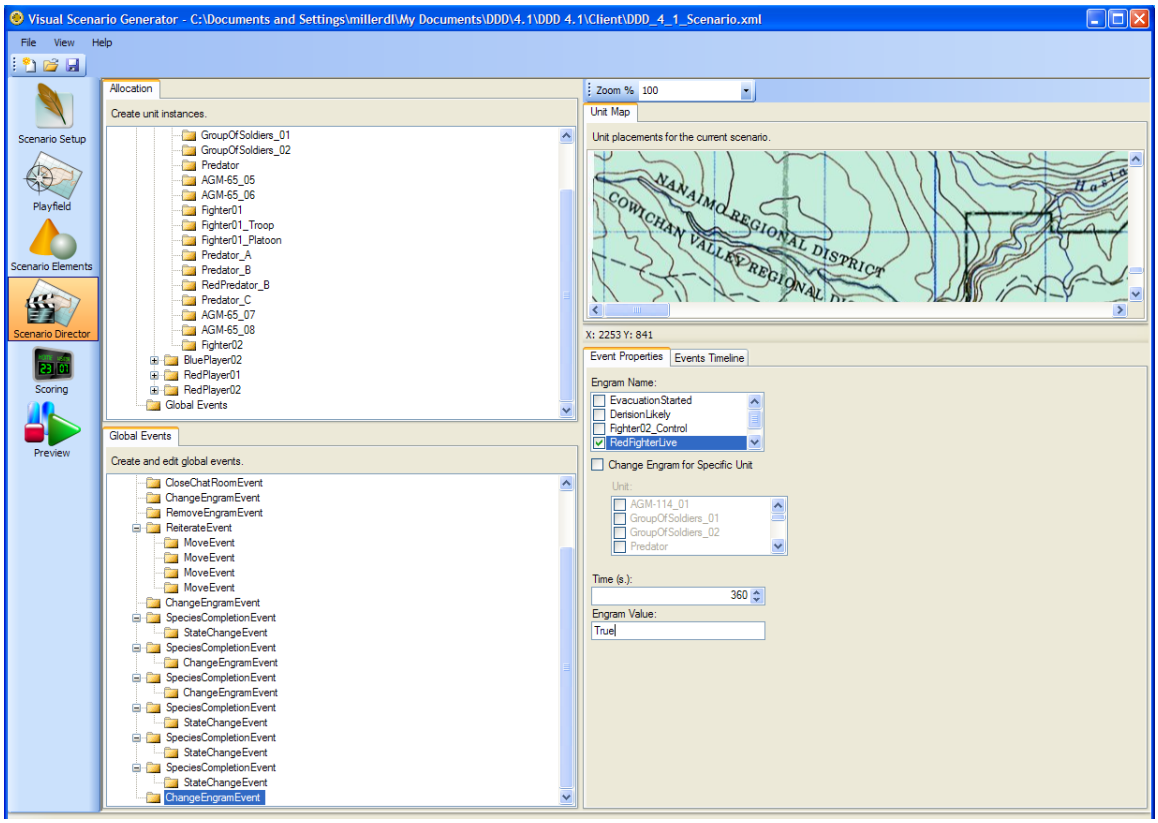


Example 6: Using Engrams

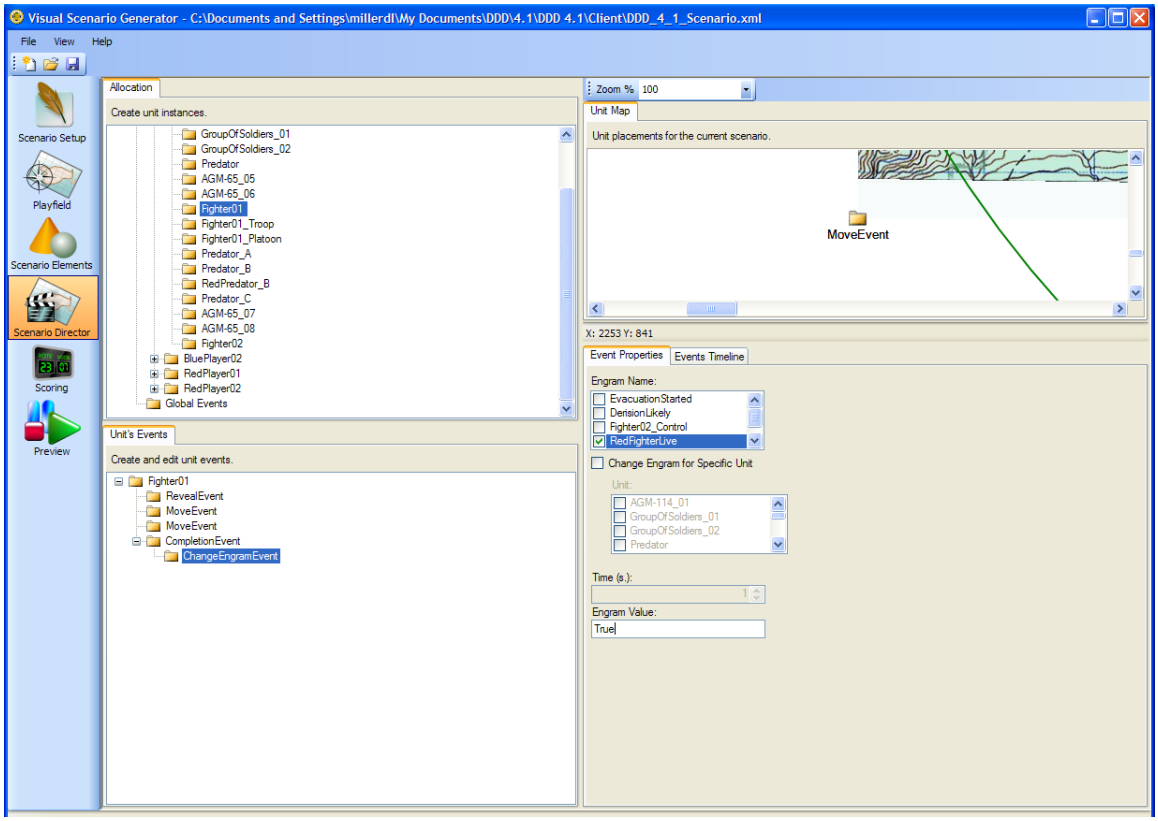
Remember that an engram is an object that is defined in the Scenario Elements view. When defining an engram, the scenario writer gives it an initial value. Engram values can be String, Double (numeric), or Logical (true or false). This example uses an engram called RedFighterLive that has an initial value of False. Using Example 5, there is already a Transfer Event that will transfer the ownership of Fighter01 from BluePlayer01 to BluePlayer02. The scenario writer can make this transfer a conditional event by specifying that it will happen only if RedFighterLive is True.



The scenario writer also has the ability to change engram values – this is a global event, because changing an engram value is not specific to one unit. Instead, changing an engram value has an impact on the entire scenario. To change the value of an engram, first select Global Events in the Allocation pane. Notice that a Scenario node is displayed in the Unit's Events pane – this is the scenario name in this example. Right click on Scenario to display its dropdown menu, as shown below. In this case, select Create ChangeEngramEvent Child. In the Event Properties pane, select a name in the Engram Name field and enter the value "Sad" in the Engram Value text box.

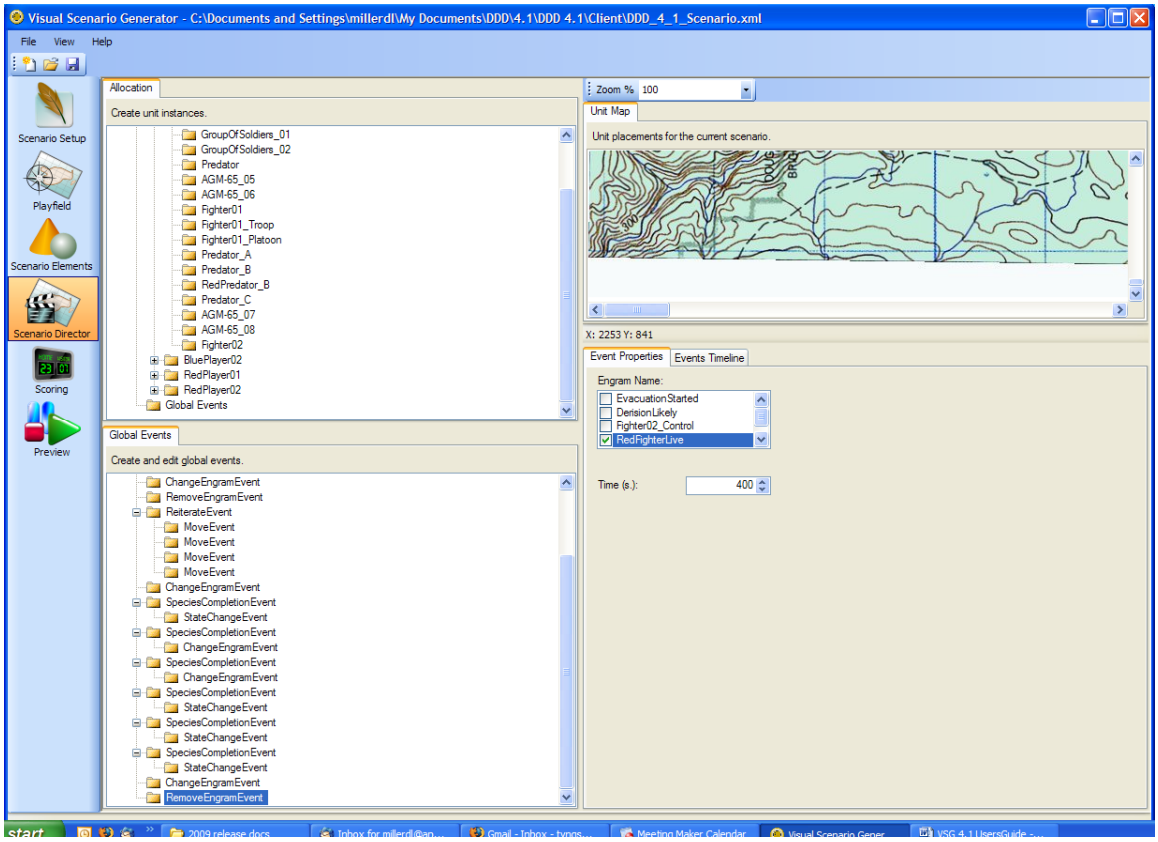

Visual Scenario Generator - C:\Documents and Settings\millerdl\My Documents\DDD\4.1\DDD 4.1\Client\DDD_4_1_Scenario.xml
File
View
Help
Scenario Setup
Playfield
Scenario Elements
Scenario Director
Scoring
Preview
Allocation
Create unit instances.
GroupOfSoldiers_01
GroupOfSoldiers_02
Predator
AGM-65_05
AGM-65_06
Fighter01
Fighter01_Troop
Fighter01_Platoon
Predator_A
Predator_B
RedPredator_B
Predator_C
AGM-65_07
AGM-65_08
Fighter02
BluePlayer02
RedPlayer01
RedPlayer02
Global Events
Global Events
Create and edit global events.
CloseChatRoomEvent
ChangeEngramEvent
RemoveEngramEvent
ReiterateEvent
MoveEvent
MoveEvent
MoveEvent
MoveEvent
ChangeEngramEvent
SpeciesCompletionEvent
StateChangeEvent
SpeciesCompletionEvent
ChangeEngramEvent
SpeciesCompletionEvent
ChangeEngramEvent
SpeciesCompletionEvent
StateChangeEvent
SpeciesCompletionEvent
StateChangeEvent
SpeciesCompletionEvent
StateChangeEvent
ChangeEngramEvent
Zoom % 100
Unit Map
Unit placements for the current scenario.
NANAIMO REGIONAL DISTRICT
COWICHAN VALLEY REGIONAL
X: 2253 Y: 841
Event Properties
Events Timeline
Engram Name:
EvacuationStarted
DerisionLikely
Fighter02_Control
RedFighterLive
Change Engram for Specific Unit
Unit:
AGM-114_01
GroupOfSoldiers_01
GroupOfSoldiers_02
Predator
Time (s.):
360
Engram Value:
True

Changing an engram value can also be based on a Completion Event. In this example, the engram value will change based on the last Completion Event of Fighter01. With Fighter01 selected in the Allocation tree, right click on CompletionEvent and choose Create ChangeEngramEvent Child After from the dropdown menu. In the Events Properties pane, as shown in the illustration below, select RedFighterLive in the Engram Name field and enter "True" in the Engram Value field.



This means that the RedFighterLive engram will change to True only when this Move Event finishes, regardless of how long that takes. In this case, changing the RedFighterLive engram is conditional.

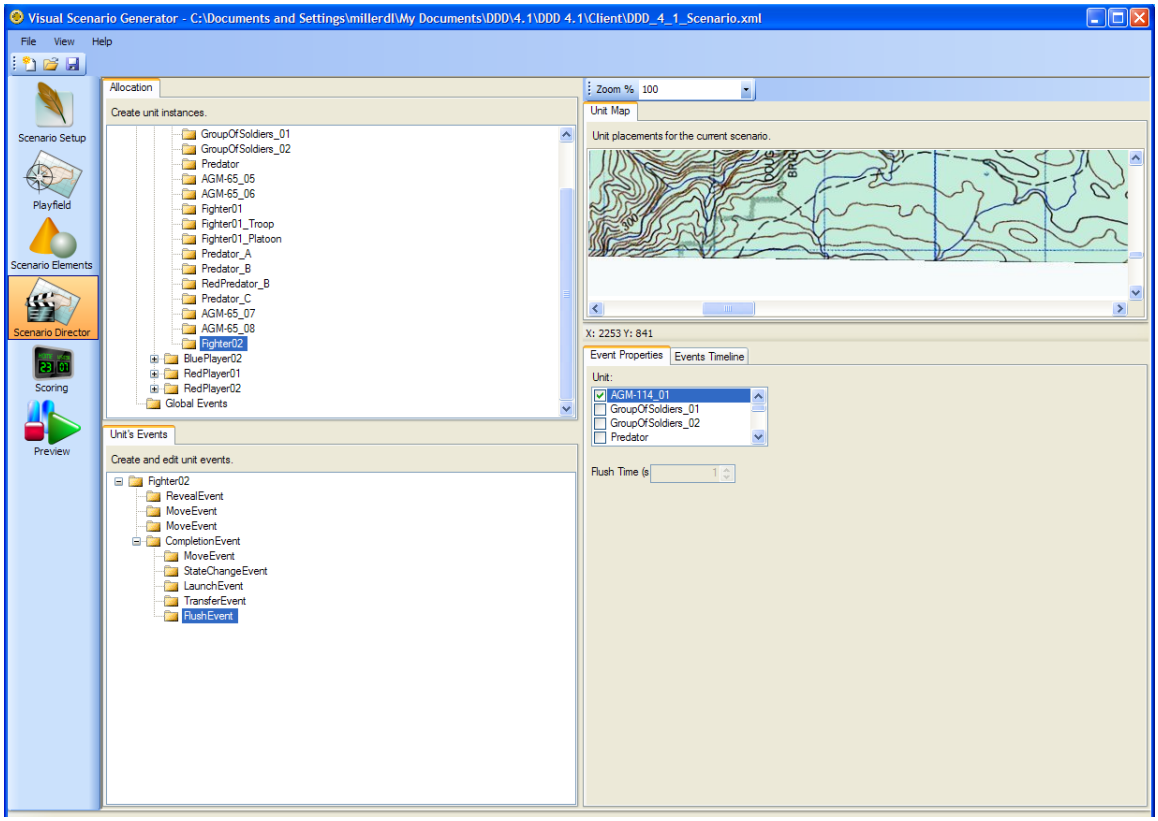
Engrams also can be made inactive. To do so, Global Events must be selected in the Allocation pane. Right click on the scenario name in the Unit's Events pane and choose Create RemoveEngramEvent Child from the dropdown menu. As shown in the following illustration, select RemoveEngramEvent so that its properties are displayed in the Event Properties pane. For this example, select RedFighterLive in the Engram Name field. Note that the Time value is 400 – this means the engram will be removed when 400 seconds have elapsed in the simulation. Any events that are conditional on the engram will not happen once the engram is removed.



Example 7: Flushing Events

Once a unit's Completion Event is triggered, it is possible to flush the rest of that unit's events by creating a Flush Event. In other words, all of that unit's future events will be removed from the queue of events. The unit will remain in its current state unless the Completion Event causes its state to change. To create a Flush Event, select it from the dropdown menu in the Unit's Events pane.

One use of a Flush Event is to create a Flush Event after a ChangeState Event that changes the unit's state to Dead. For example, if Fighter02 enters the Dead state before its subplatform is launched, a Flush Event will prevent Fighter02's subplatforms from being launched. AGM_114-01 is selected in the Unit field – this means AGM_114-01's events will be flushed but the events for other units in the list will remain intact. The Flush Time indicates the time relative to the ChangeState Event at which all events for the selected unit will be flushed.

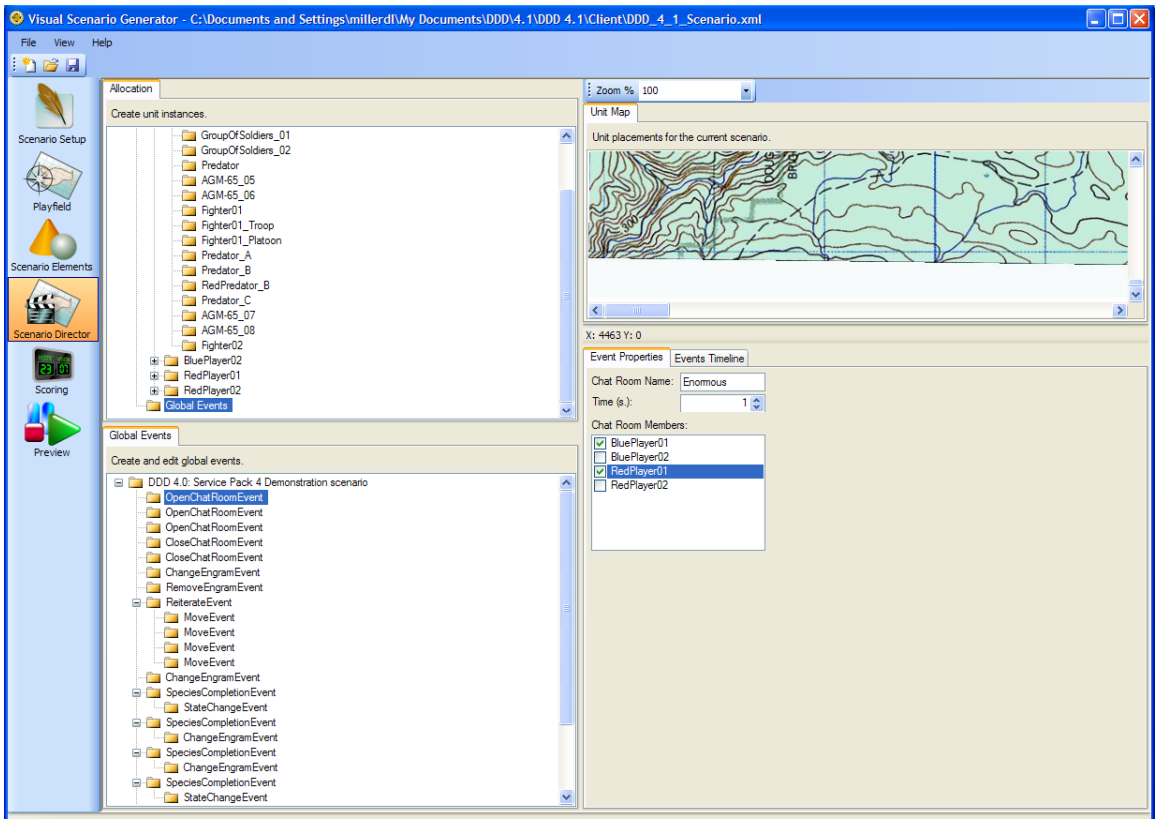


Example 8: Using Global Events

Other than the Engram and Flush events already described, the scenario writer can create the following global events:

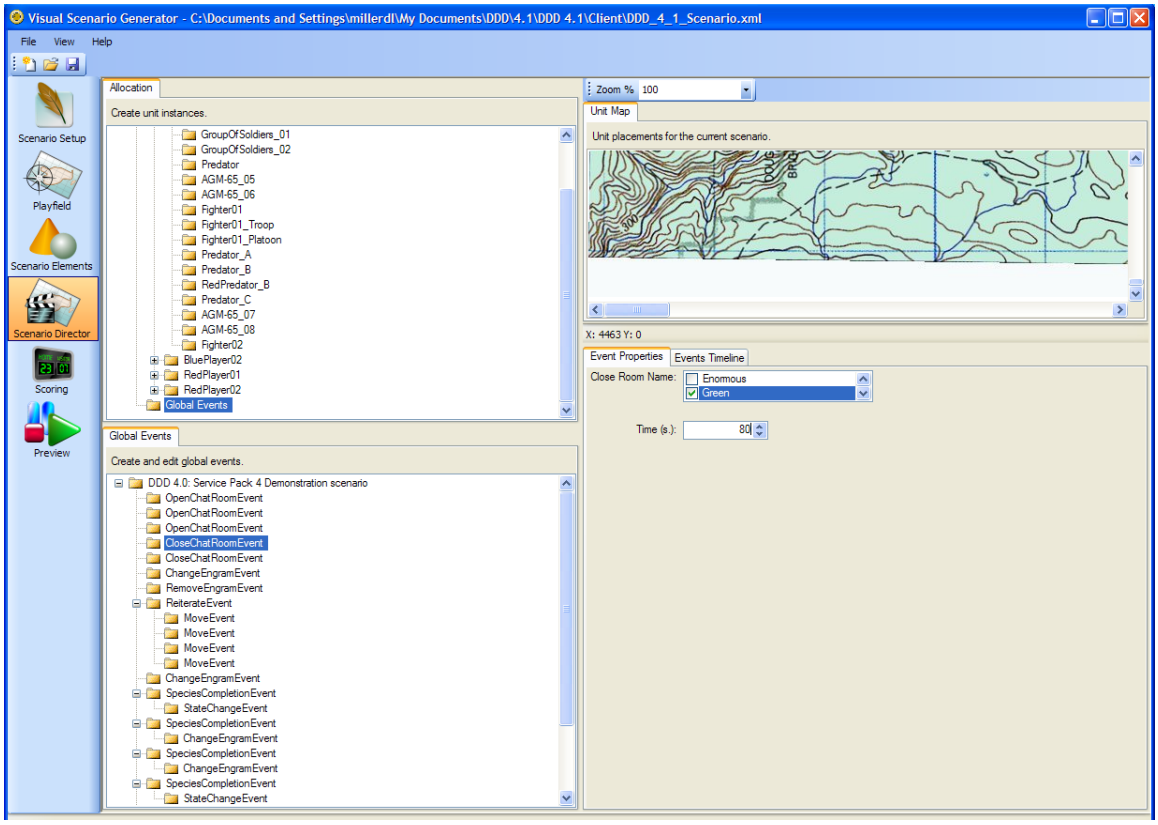
- OpenChatRoom
- CloseChatRoom
- OpenVoiceChannel
- CloseVoiceChannel
- SendChatMessage
- OpenVoiceChannel
- CloseVoiceChannel
- SendVoiceMessage
- SendVoiceMessageToUser
- Reiterate
- SpeciesCompletion

Chat rooms can be created to provide a way for specified Decision Makers to share information. To open a chat room, first select Global Events in the Allocation pane. Right click on the scenario name in the Unit's Events pane and choose Create OpenChatRoomEvent Child from the dropdown menu. When the new OpenChatRoomEvent node is selected, its properties are displayed in the Events Properties pane, as shown below.



Enter a name for the chat room in the Chat Room Name box. The Time value indicates the time in the scenario at which the chat room will be available to Decision Makers. The default time of 1 means that the chat room will be available at one second after the simulation begins. Select the Decision Makers that have access to the chat room in the Chat Room Members field. The scenario writer can create multiple chat rooms with different members.

To close a chat room, right click on the scenario name and choose Create CloseChatRoomEvent Child from the dropdown menu. In the following illustration, Green is the chat room that will be closed by this event. The Time value indicates that the Green chat room will close 80 seconds after the simulation begins.

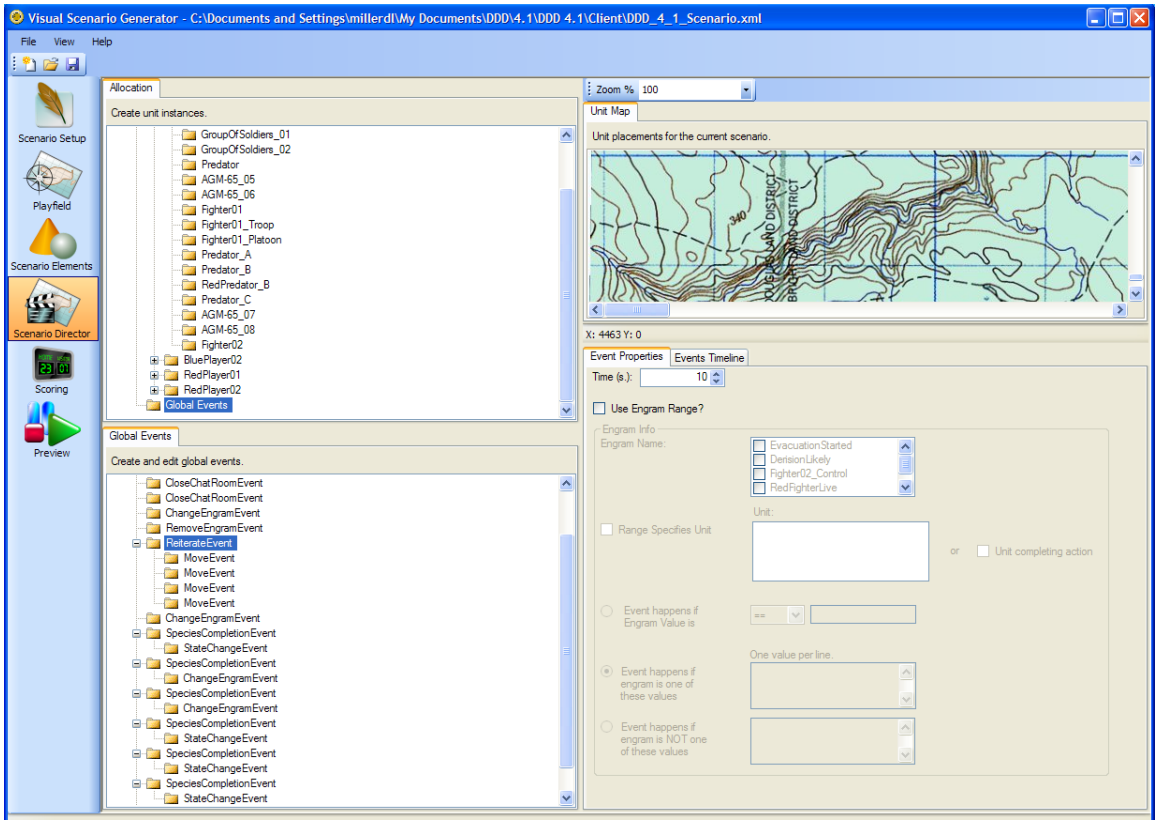


The SpeciesCompletion Event is very similar to the Completion Event for a unit. However, instead of completing an event for a specific unit, the completion event is for an entire species of units. For example, Fighter01 is a unit that is created from the F16-C Species Definition. Other Fighters are created from the same F16-C Species Definition. A SpeciesCompletion Event will impact Fighter01 and any other Fighters as well.

For example, a SpeciesCompletion Event can specify that when any fighter changes to the Disabled state, it will move to a certain area on the map. It is also possible to specify that if any fighter changes to the Disabled state, then an event for another unit is triggered.

To create a SpeciesCompletion Event, right click on the scenario name in the Unit's Events pane and choose Create SpeciesCompletionEvent Child from the dropdown menu. In the Events Properties pane, select the name of a species that has already been defined, and then specify an action or state. This action or state will affect every unit that was created by using the selected Species Definition.

Use a Reiterate Event to put a series of Move Events in a loop, as shown in the illustration below.



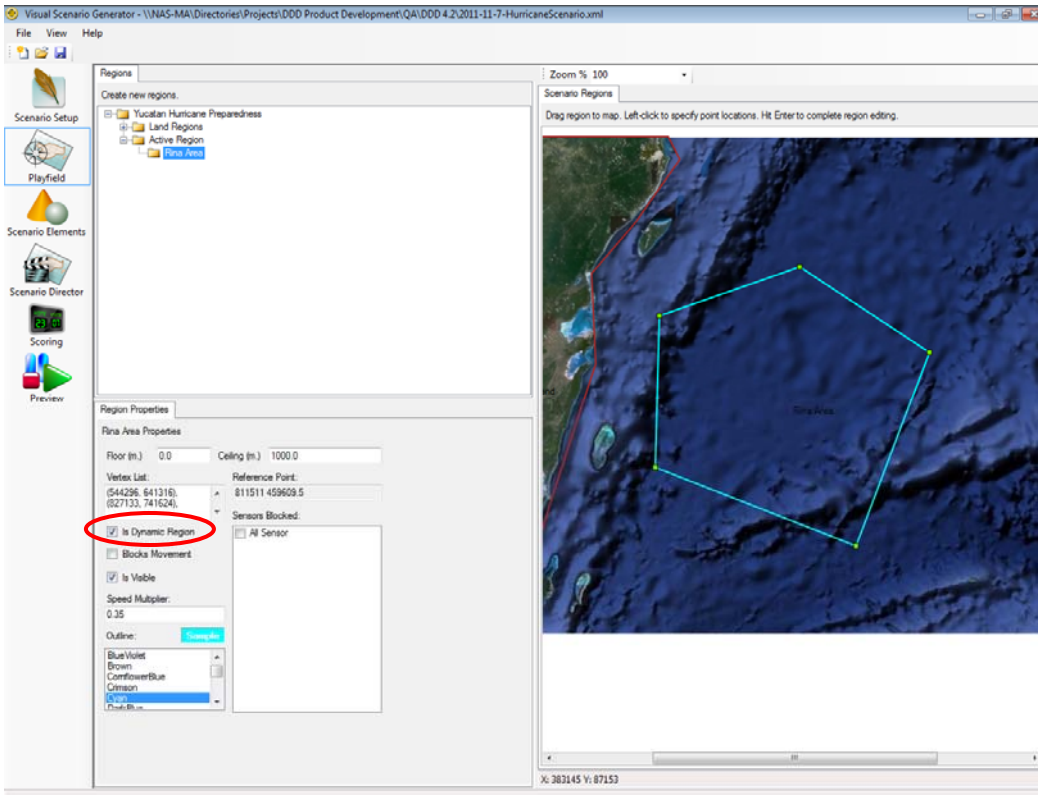
For example, Fighter01 can be made to move in a specific pattern and keep moving in that pattern during the course of the simulation. For a more specific example of using the Reiterate Event, see "How Can I Make an Asset Travel in a Continuous Pattern?".

To create a Reiterate Event, right click on the scenario name in the Unit's Events pane and choose Create ReiterateEvent Child from the dropdown menu. Then create Move Events as children of the Reiterate Event.

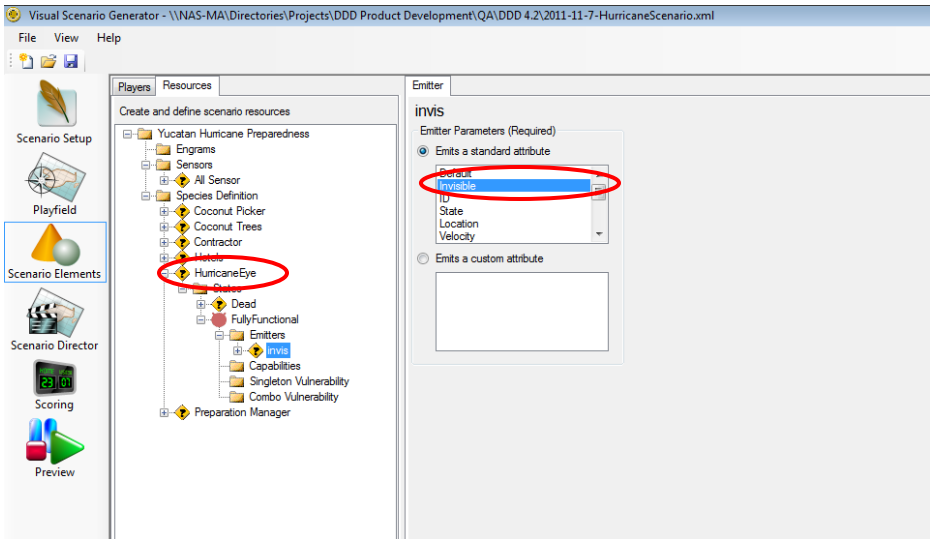
Example 9: Defining a Dynamic Active Region

The use of dynamic active regions enables the scenario writer to mimic changing environmental patterns (e.g., weather, wildfire) and create corresponding obstructions and sensor blocks. Defining a dynamic active region is a multi-part process, as described in the hurricane example below.

In the Playfield view, the scenario writer creates an active region and specifies it as dynamic by clicking the "Is Dynamic Region" checkbox in the Region Properties tab as shown in the figure below.

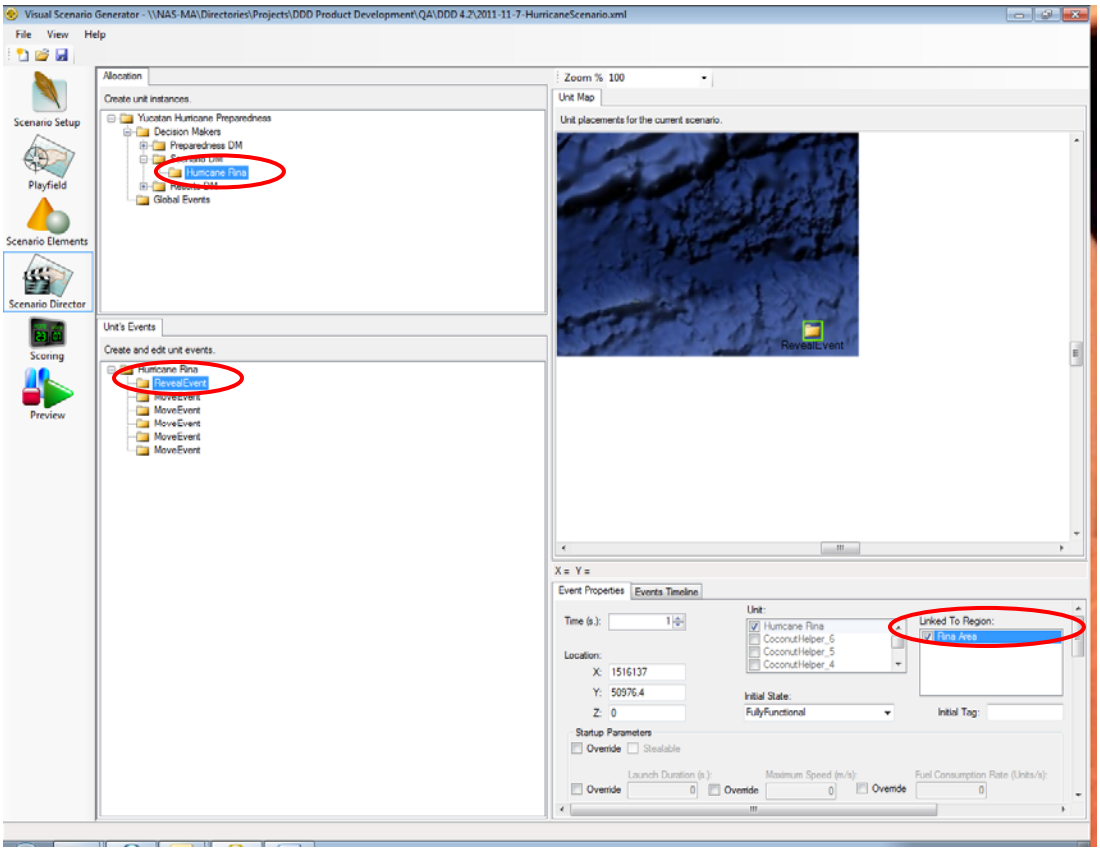


In the Scenario Elements view, the movement of the region is defined by creating an object capable of movement (e.g., HurricaneEye in this example). Optionally, the object's emitters can be specified as invisible to suppress display of the object in the DDD client views. To the player, the region will appear to move without the underlying object being shown.



Once both the dynamic active region and the corresponding object have been defined, the object is instantiated in the Scenario Director view (Hurricane Rina in the example below). Then, a

series of reveal and move events are defined that specify the behaviors for the linked active dynamic region (Rina Area in the example below).



9.0 Frequently Asked Questions

The following examples answer frequently asked questions.

What is "Absolute Time"?

The VSG user interface and user documentation refers to "Absolute Time" in the context of event execution in the Scenario Director View.

When an event is set to occur in "Absolute Time", it is meant that an event is guaranteed to execute at a particular time, without regard to state information (other than Dead), or the timing and execution of other events.

For example, A ***MoveEvent*** on ***Unit1*** to location X, Y at time 50 will always occur at time 50 (as long as the Unit1 isn't in already in a Dead state).

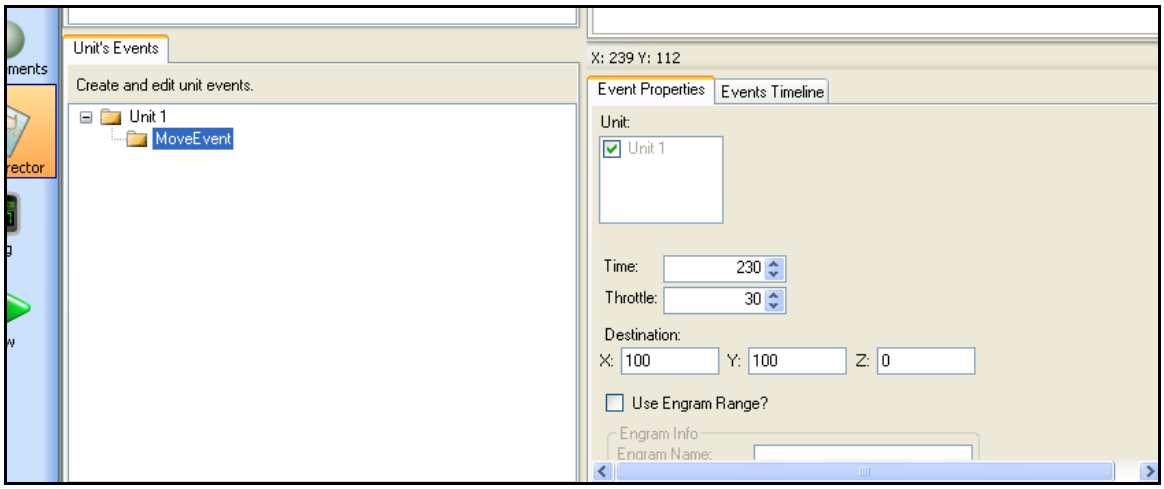
Contrast this with an event that occurs in "Relative Time", in other words, an event that exists within the context of a ***CompletionEvent***. The time at which this event will occur will be relative to the completion of the *Action* or *State Change* defined by the ***CompletionEvent*** to which it belongs.

For example, A ***MoveEvent*** to location (x,y) *OnCompletion* of the *Partially Functional* state change. The time at which the move event occurs, is relative to the time the "*Partially Functional*" state change completes.

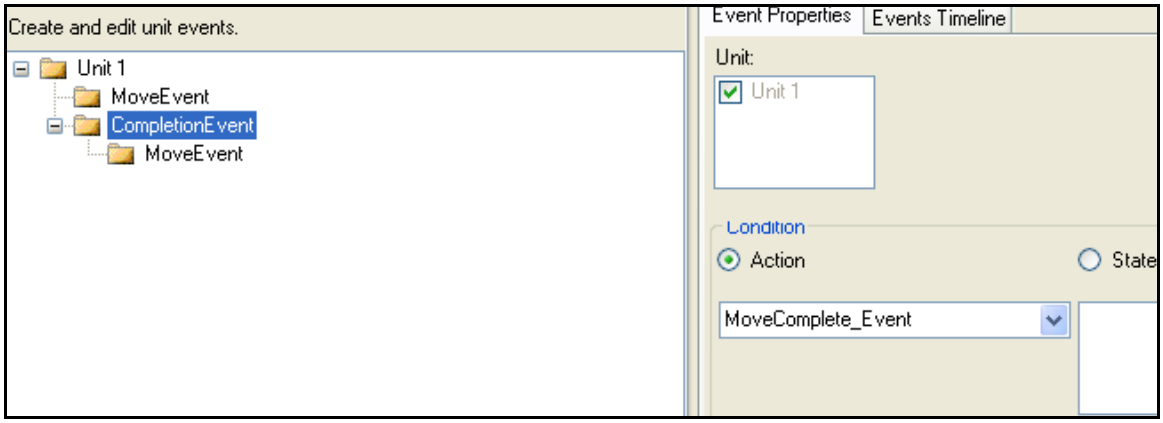
How Can I Make an Asset Travel Along a Path?

The logic of having a unit move through a series of waypoints is to explicitly cause it to make the first leg of the trip, and then create a series of commands to indicate that when the unit reaches its first destination, it should start moving toward the next.

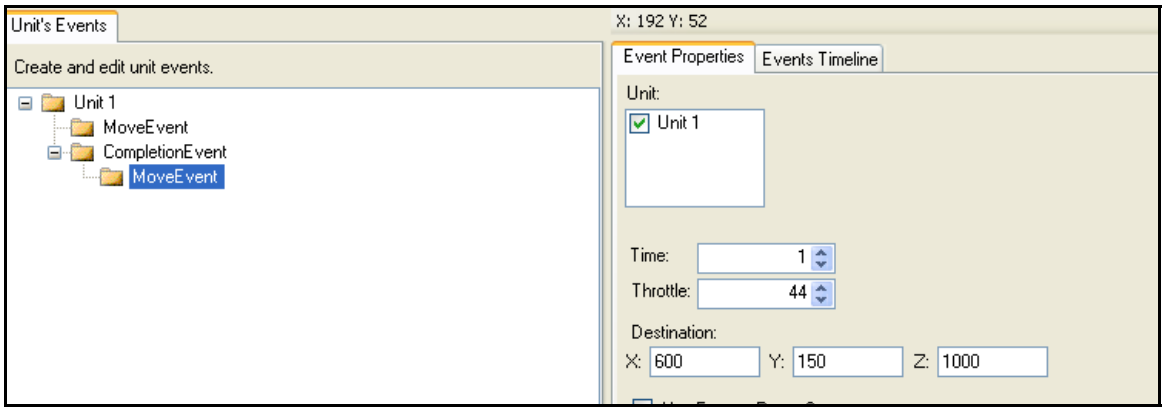
A path with one intermediate point might be defined as follows. First, create the initial Move Event, as illustrated below.



Next, create a Completion Event.



The second leg of the trip is defined in the child Move Event, whose properties are shown below.



Notice that there is no reference in the Completion Event definition to the previous Move Event. The Completion Events in the scenario are processed strictly in the order they appear! Once a Completion Event has been processed, it is removed from future consideration, so a path with an arbitrarily long sequence of waypoints can be specified. Note that once a unit has been destroyed (moved to the Dead state) all future Completion Events involving it will be deleted.

The Completion Event mechanism is intended for use only with units that are not assigned a player. If there are Completion Events that name units controlled by live players, their behavior may be unpredictable as soon as the players begin to manipulate them.

How Can a Unit Be Made to Reach Certain Waypoints at Specific Times?

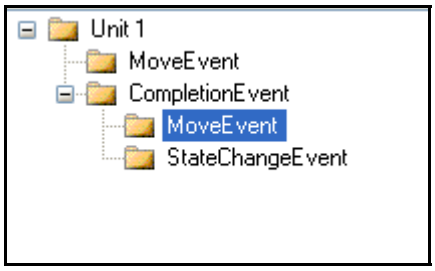
If it is desired to have the unit reach certain waypoints at specified times, it will be necessary to calculate the throttle settings that will make this happen. For example, suppose a unit has a maximum speed of 66 km/hour (18.33 meters/second), and that it is required to start moving at time t=10 from the point (0,0), to reach the point (0, 5280) after eight minutes, and then to reach the point (4000, 12000) ten minutes later. The initial Move Event command must specify a throttle setting of 60%: at a throttle setting of 60% the unit travels at a rate of 11 meters/second, and will require 480 seconds to reach the first waypoint. The throttle setting for the second leg of the trip will be 0.71: the distance is 7820.38 meters and to travel that distance in ten minutes requires a speed of 13.03 meters/second, which is 71% of the maximum speed.

This kind of precise timing cannot be guaranteed during execution of the scenario. If there are other events specified for this unit that happen during the time it is supposed to be moving, they may interfere with the position or speed of the unit. Similarly, an external event, such as an

attack that causes the state to change to one with a different maximum speed, may also affect the timing.

Can There be More Than One Event Specified in a Completion Event?

Yes – more than one event can be specified in a Completion Event. Each is shown as a child of the Completion Event.



Here, the StateChange Event might have been added in one of two ways:

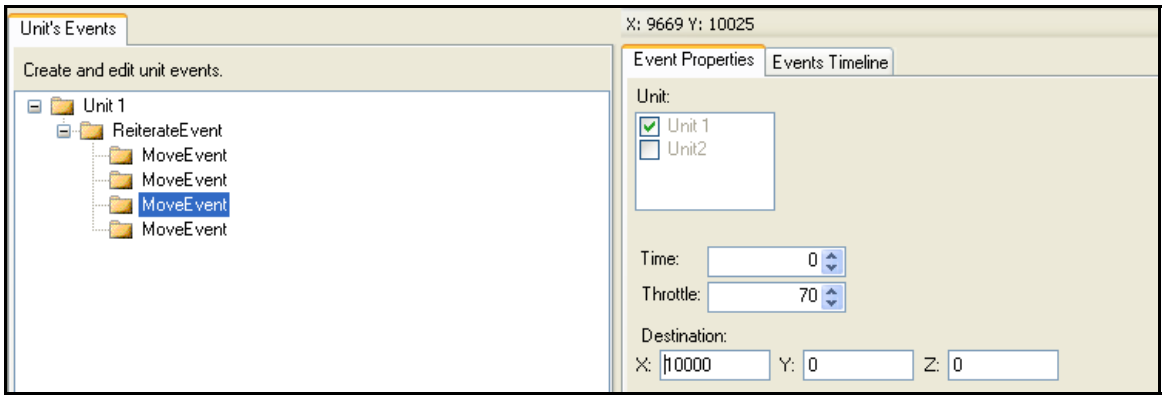
- Right click on CompletionEvent and select Add MoveEvent Child.
- Right-click on MoveEvent and select Add StateChangeEvent After.

Can More Than One Unit Be Referred to in a Completion Event?

Yes -- any unit can be made to act contingent upon any event occurring. The previous example can be changed by adding a second unit and a Move Event for it that is conditioned on the same Completion Event.

How Can I Make an Asset Travel in a Continuous Pattern?

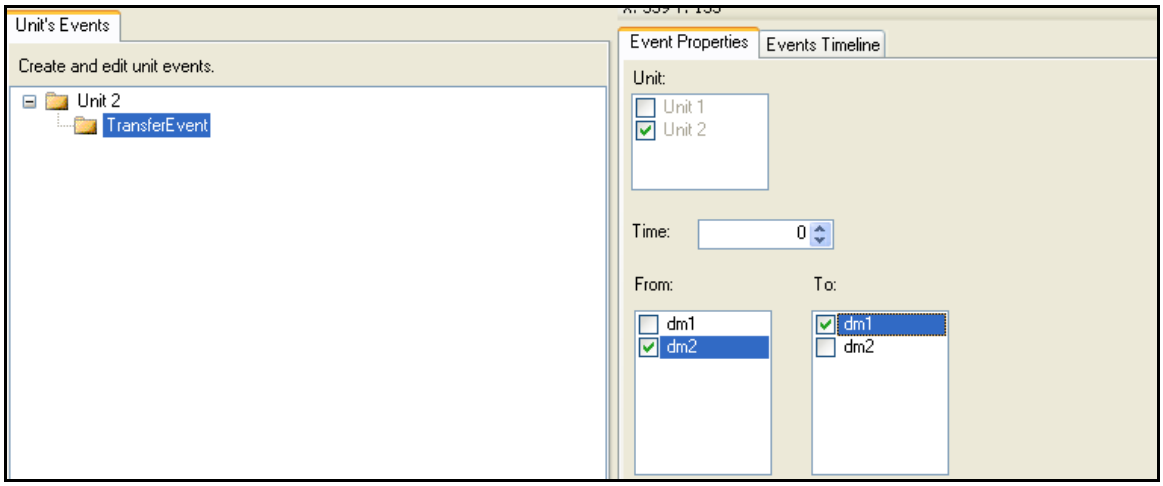
While Completion Events can be used to create an arbitrarily long path for a unit to follow, they are not the appropriate way to cause an infinitely repeating pattern of moves. For this situation, use the Reiterate Event. In this example, the four Move Events have respective destinations of (0, 10000), (10000, 10000), (10000, 0) and (0, 0), and all happen to have the same throttle setting. Assuming that the unit was Revealed at (0, 0), this command will cause it to continually move along the perimeter of a square at a constant rate.



With the one difference that the commands are continually repeated, the Reiterate Event shares all the constraints of the Completion Event. One additional constraint is that nearly any command can go in the DoThis section of a Completion Event, but only a Move Event can be specified in a Reiterate Event. Notice, also that the Time fields in the individual Move Event commands are ignored. Once the sequence begins, at the time specified in the Time field of the Reiterate Event it runs inexorably. The Time fields in the individual Move Events are all ignored.

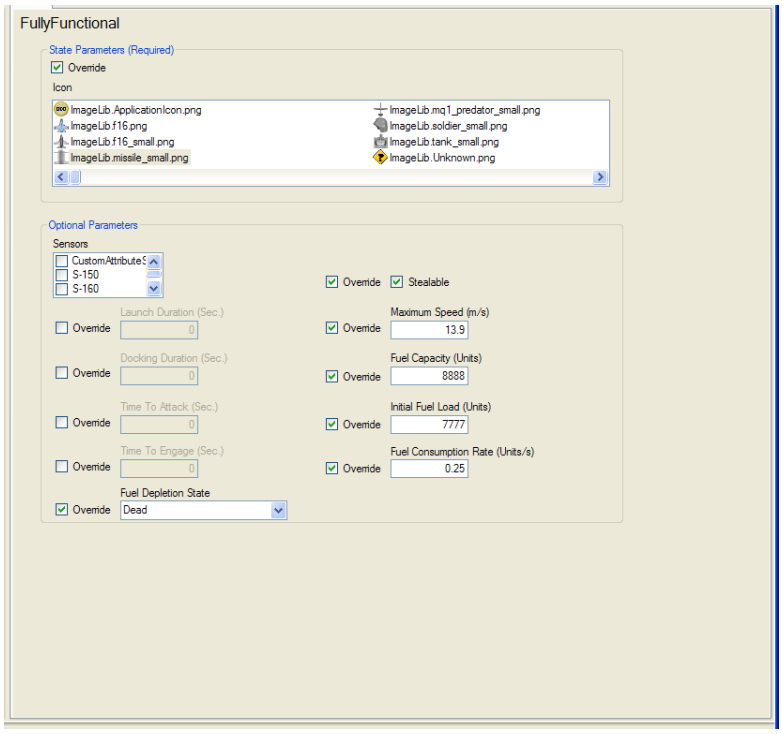
How Can Units Be Transferred Between Decision Makers? What Kinds of Transfers are Possible?

Every unit belongs to some Decision Maker. In the scenario, it is possible at any time to have a command that transfers a unit from one Decision Maker to another. Here is an example screen:



This example specifies that when the Transfer Event command is executed, Unit 2 will be transferred from dm1 to dm1. If Unit 2 does not belong to dm2 when Transfer Event is executed, nothing will happen.

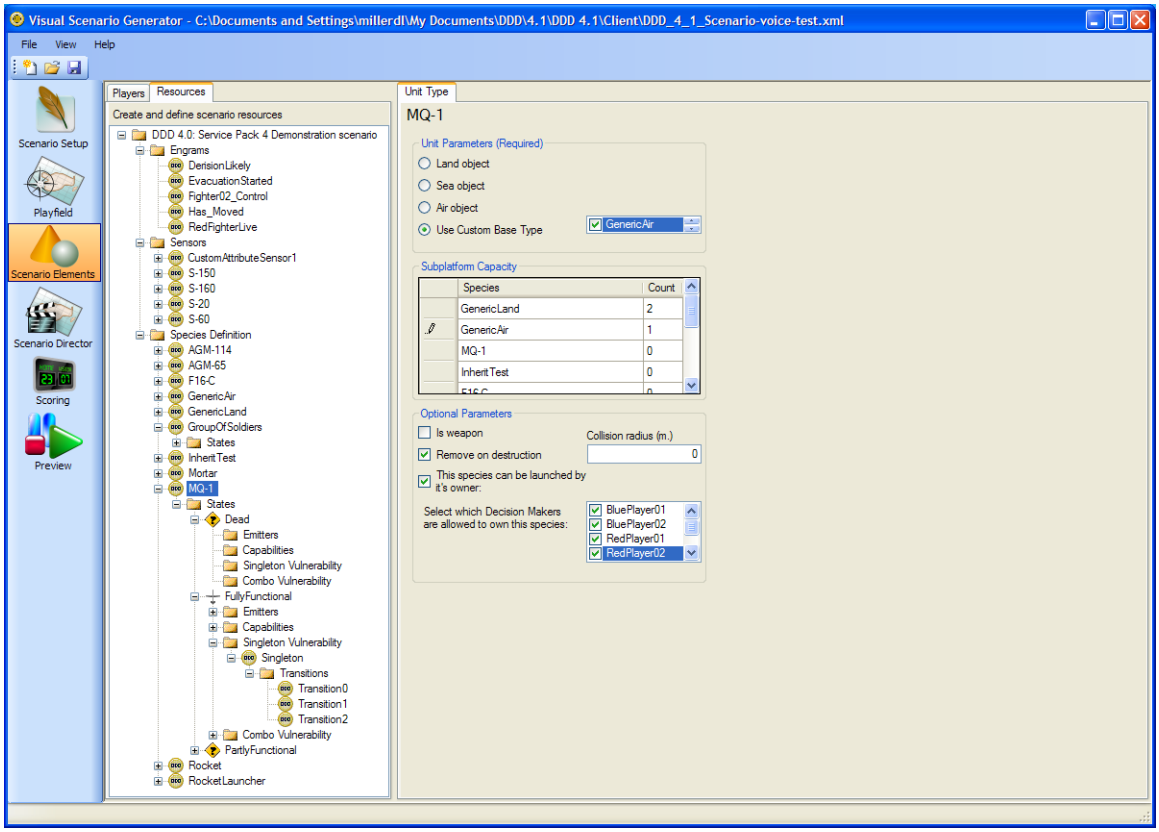
It is also possible for a player to transfer an asset during a game. Typically, only the owning Decision Maker can initiate a transfer. However, you can define a unit in a way that lets any player transfer it. This can be done in a species definition as shown in the next figure. Setting the parameter Stealable makes it possible for any player to initiate a transfer. This parameter can have different values for different states and can also be assigned when a unit is Revealed.



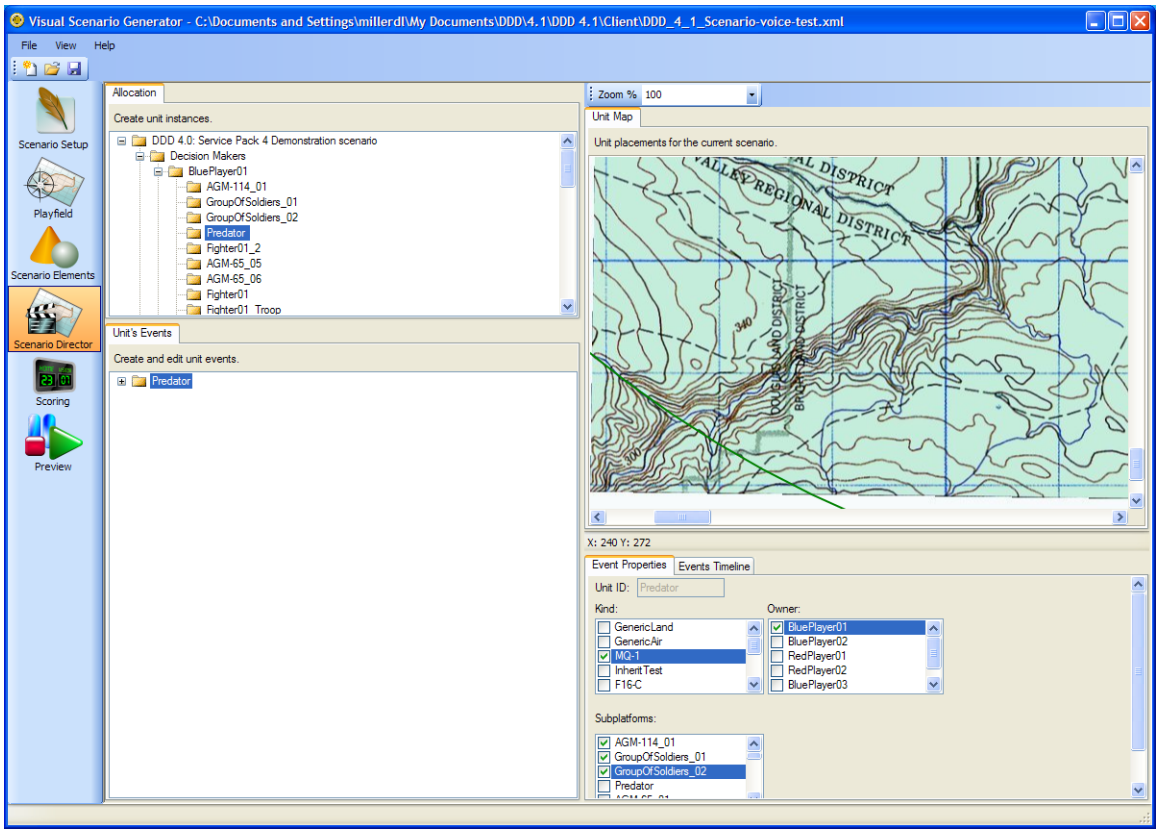
Subplatforms need not belong to the same Decision Maker as their parent. Transferring a subplatform has no effect on the ownership of the parent. When a parent unit is transferred, the docked subplatforms are transferred at the same time, but the active ones are not.

How Can I Put Subplatforms on an Asset?

When an species is created, it is possible to indicate that it can have subplatforms docked to it – think of F/A-18 Hornets on a carrier, M1-tanks on an LST, or grenades on a soldier. Remember that before subplatforms can be defined, a Decision Maker and a Species Definition must already have been defined in the Scenario Elements view. In this example, MQ-1 is a GenericAir object that can include Subplatforms as specified in the Subplatform Capacity table – in this example, 2 species of type GenericLand and 1 of type GenericAir.

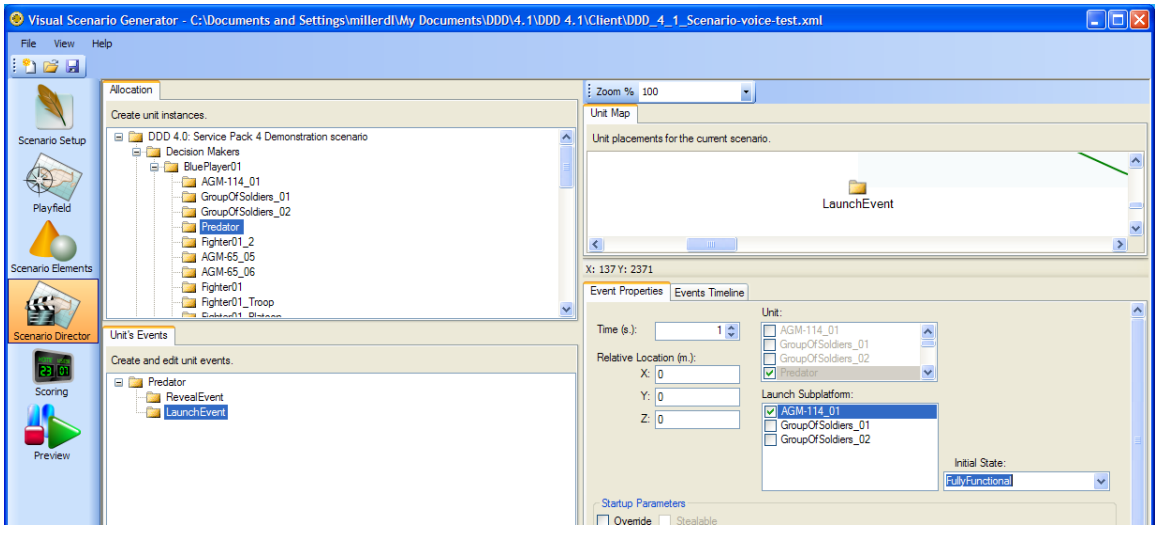


In the Scenario Director view, "BluePlayer01" is the Decision Maker and several species have been defined including "Predator" (a unit of type MQ-1), "AGM-114_01", "GroupOfSoldiers_01", and "GroupOfSoldiers_02". Since units of type MQ-1 were defined to allow subplatforms, when a CreateEvent is established for Predator, a set of subplatforms can be selected for it as shown below.



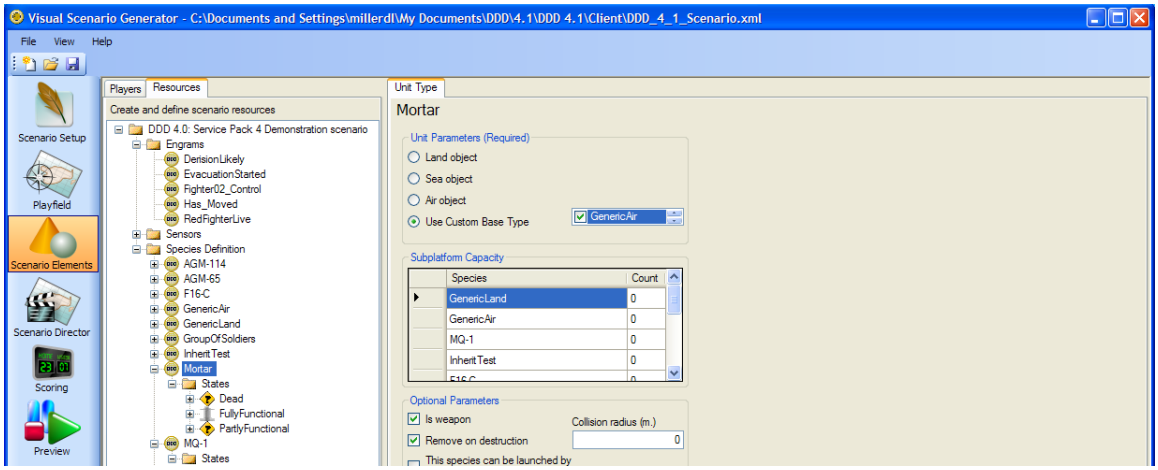
How Can Subplatforms Start Up as Active Assets?

To use subplatforms as active assets at the onset of the simulation, create a LaunchEvent for the subplatform at Time equal to 1 as shown below.

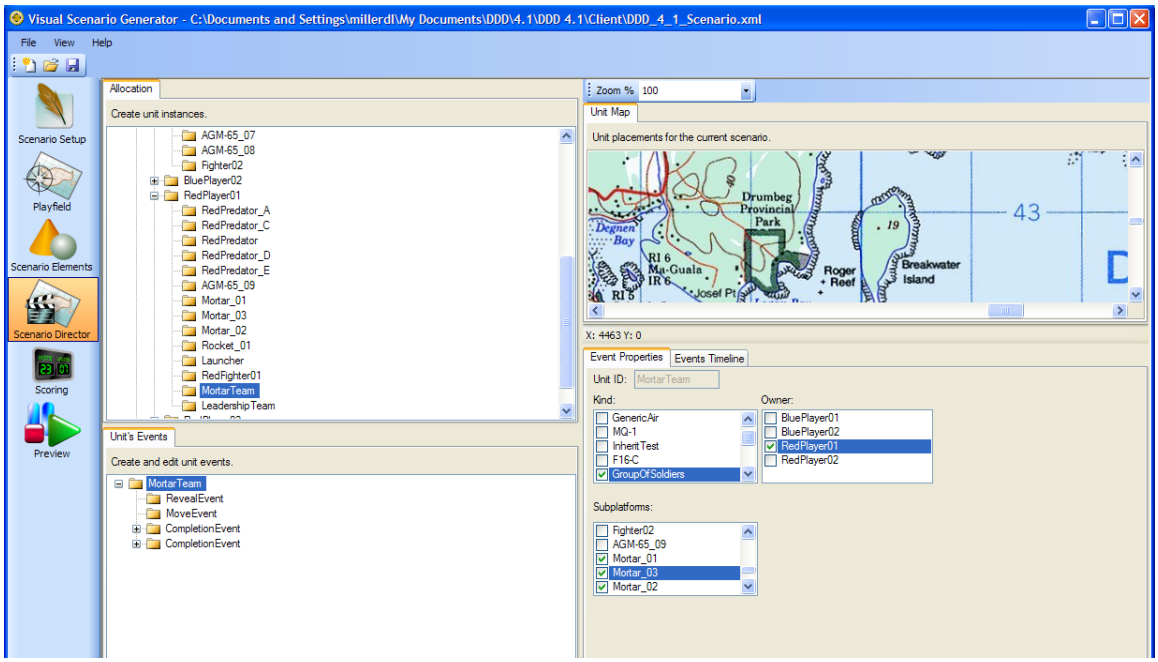


How Can a Unit be Given Weapons?

To define a species as a weapon, check the Is weapon field in the Optional Parameters field of the species definition, as shown below.



To place weapons on a unit, simply list them as subplatforms in the definition of the unit. This can be done by defining them to be docked subplatforms, as shown below.



How Can a Subplatform of a Unit be Given Weapons?

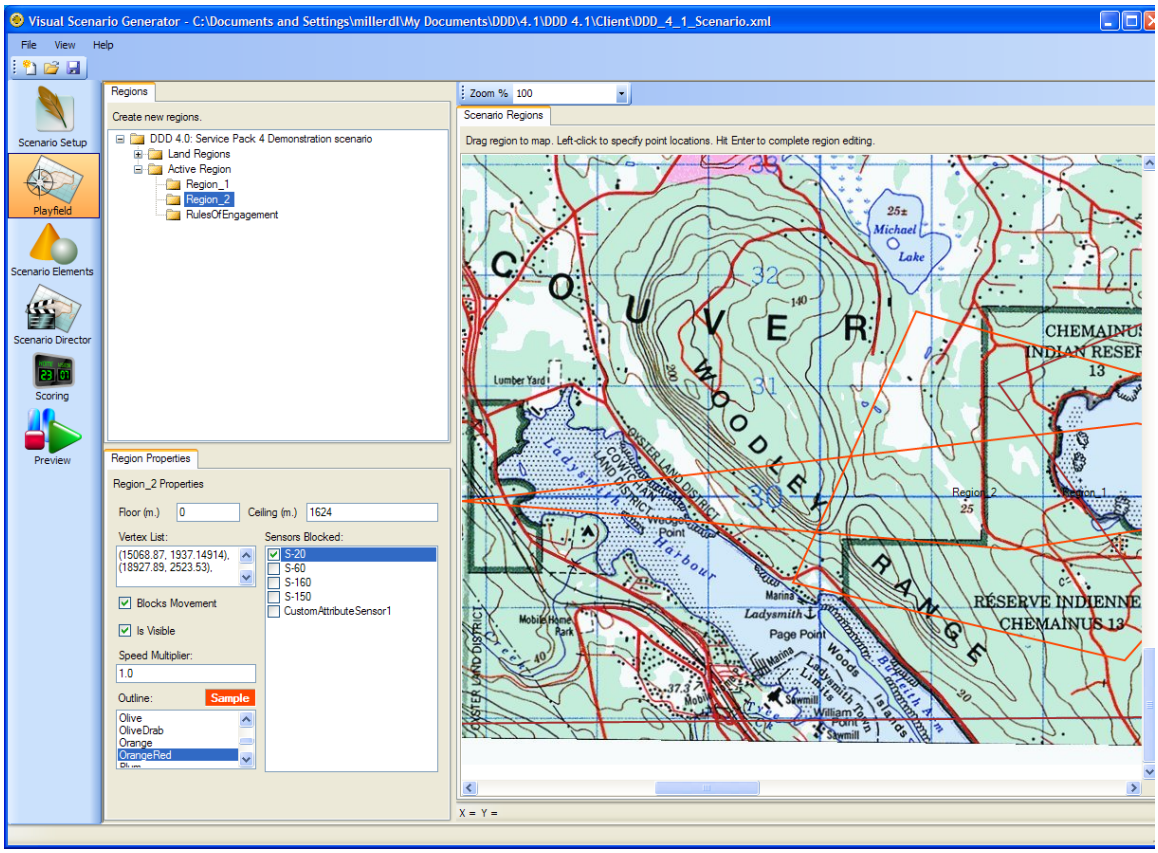
Weapons can be allocated to a Subplatform using the same process as described above for units, since subplatforms are units themselves.

How can Subplatforms be Given Subplatforms of Their Own?

Subplatforms can be allocated to a "parent" Subplatform using the same process as described above for units, since subplatforms are units themselves.

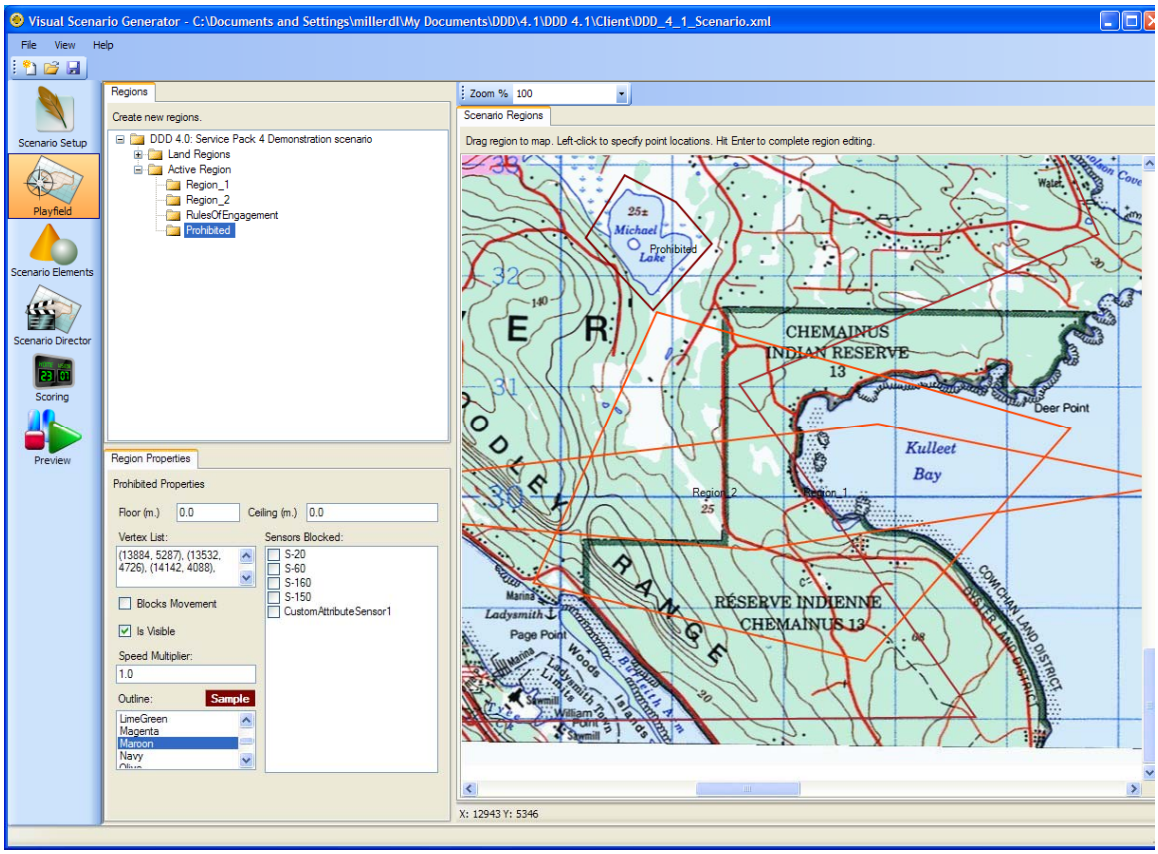
How Can I Create a No-Fly Zone?

There are different ways to create a no-fly zone, depending what the scenario writer needs. In the figure below, Region_2 is an Active Region that has the "Movement Blocked" feature turned on. This is a no-fly zone because it completely blocks all units from passing through it (up to an altitude of one mile). Also notice that this active zone is a 3D zone with a Start at the ground level and a ceiling at 1624 meters.

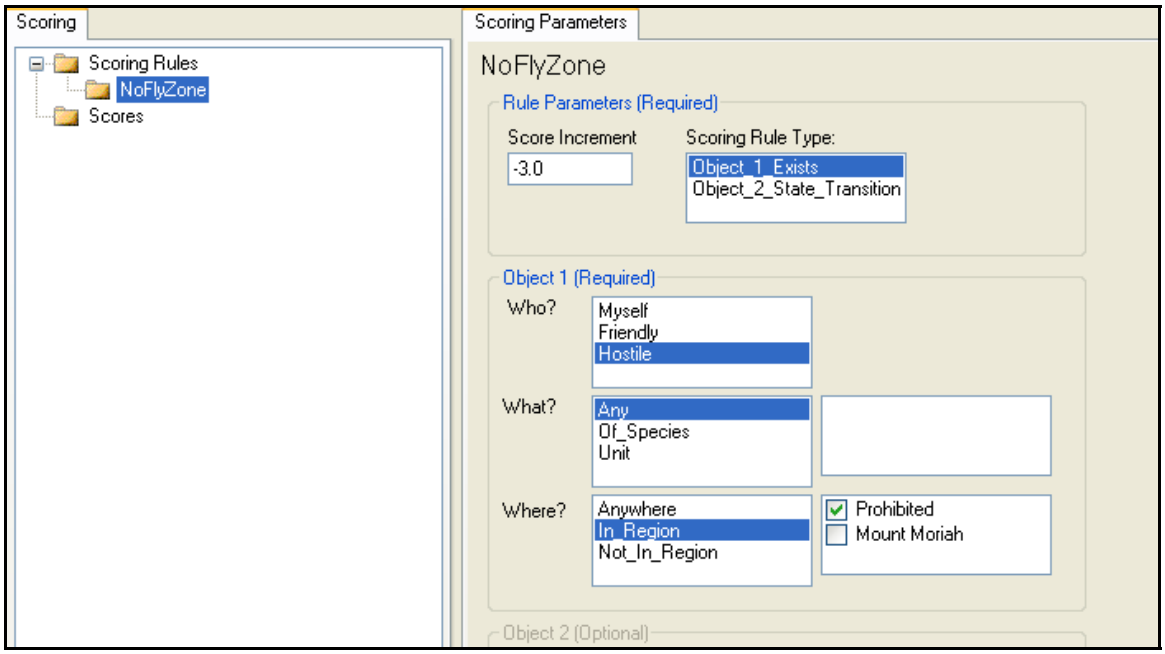


In some cases a no-fly region is one in which travel is not impossible but merely prohibited. That is typically implemented by attaching a negative score to travel within it. For example, a goal could be to have a particular BlueForce decision-maker lose three points whenever a RedForce unit is in the zone.

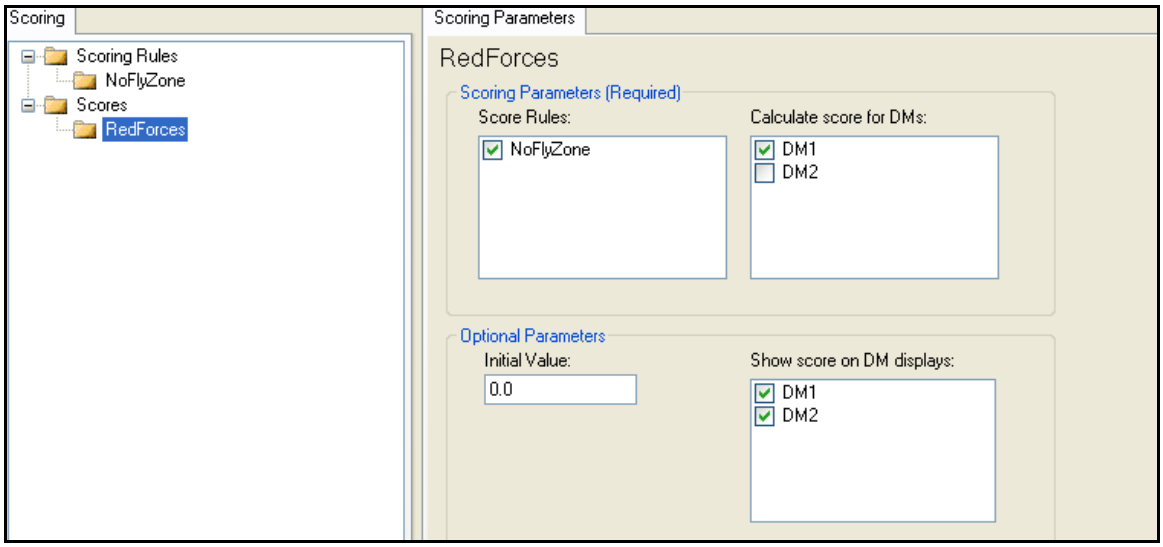
The region named Prohibited in the next figure can be made into such a no-fly zone, but there is nothing about the definition of the region itself that does this.



Instead, it is necessary to create a scoring rule based on the region. The next figure shows such a scoring rule. It specifies a loss of three points for any hostile unit that travels inside the region Prohibited. Note that this score decrement will be applied for every hostile unit that is in the region, and it will be subtracted every second that a unit is in the zone. Note, too, that the rule does not specify from whose viewpoint hostility is to be interpreted.



Scoring rules are not complete without the context of a score. For example, the NoFlyZone rule appears in the following score definition:



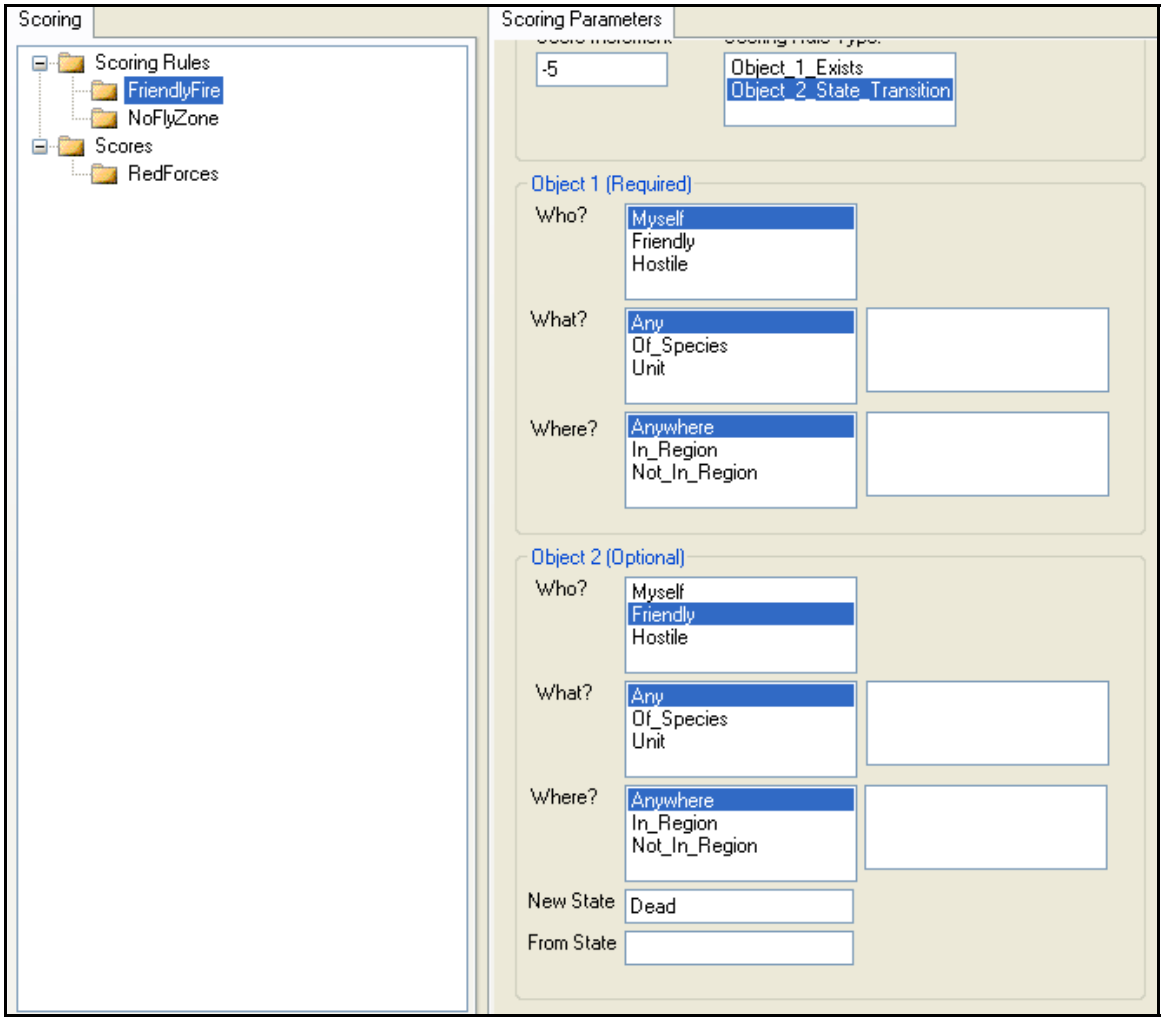
The complete interpretation is that during every second, every unit in the playfield is examined. If the unit is hostile (from the rule) to DM1 (from the score definition), then DM1 loses three points. The figure also shows that both players can view the score.

How Can I Make Damage from Friendly Fire Possible?

If a team is listed as one of its own opposing teams, then units belonging to members of the team will be able to attack one another.

How Can I Penalize Friendly Fire?

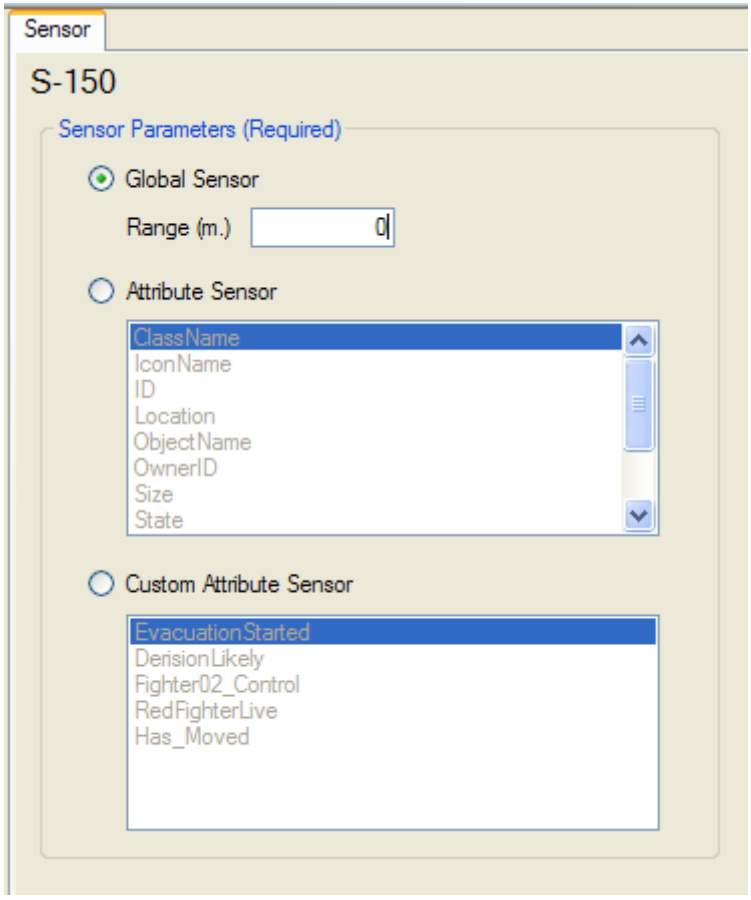
The term "friendly fire" is interpreted as meaning the destruction of a unit that is friendly to the unit getting the kill. The rule definition below is an instance.



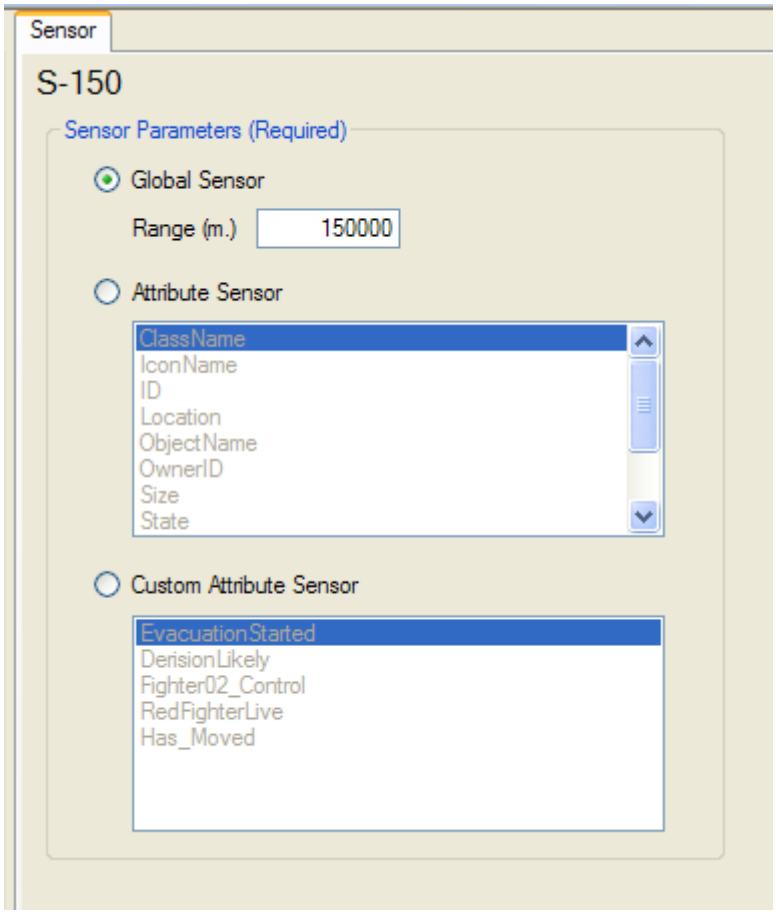
Remember that the rule will be used in conjunction with one or more Scores, each of which names some collection of Decision Makers. Taking each such decision Maker as being "Myself," this rule specifies that if any unit belonging to the Decision Maker causes a State Transition of a friendly unit into the Dead state, each score containing this rule loses five points.

How Can Sensing Abilities Be Made to Depend on Distance?

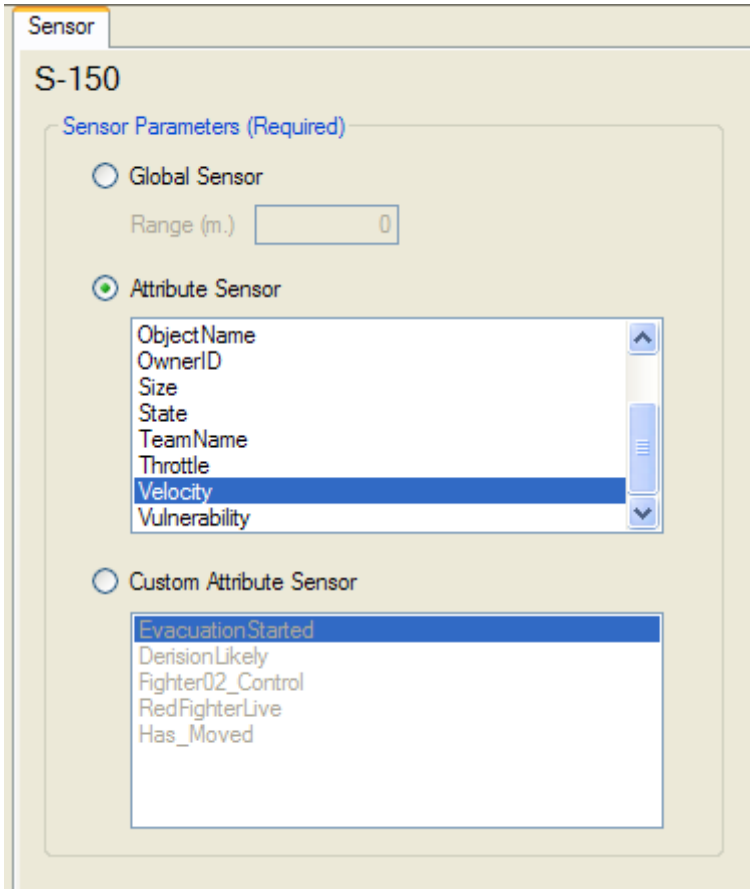
The easiest way to define a sensor is to leave the parameters set to the defaults.



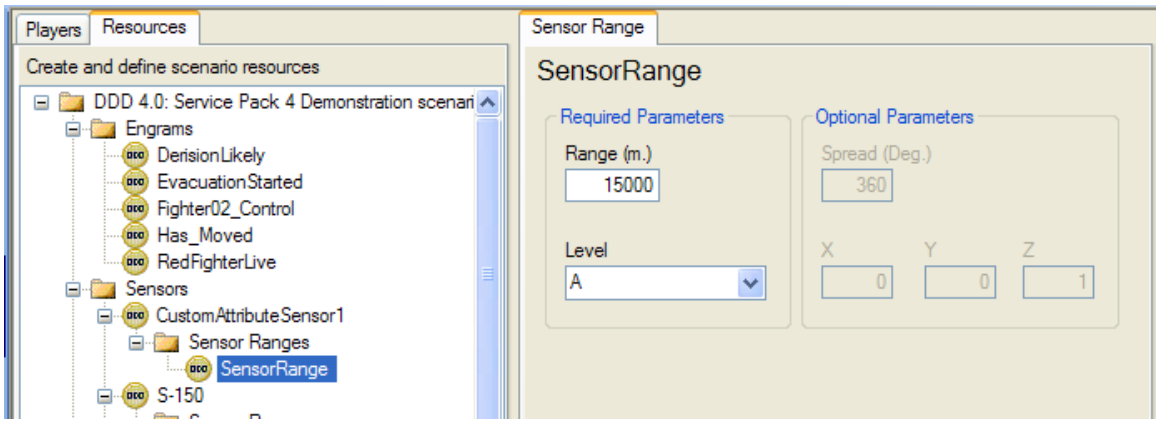
The value of 0 for the range indicates that the sensor's range is unlimited. Inserting a non-zero value for the range would limit to sensor to that number of meters–150 kilometer in the following example.



An alternative is to limit the sensor to a single attribute. Note that an asset can carry many sensors, each sensitive to a different attribute. In the following figure, the sensor is defined to detect only velocity.

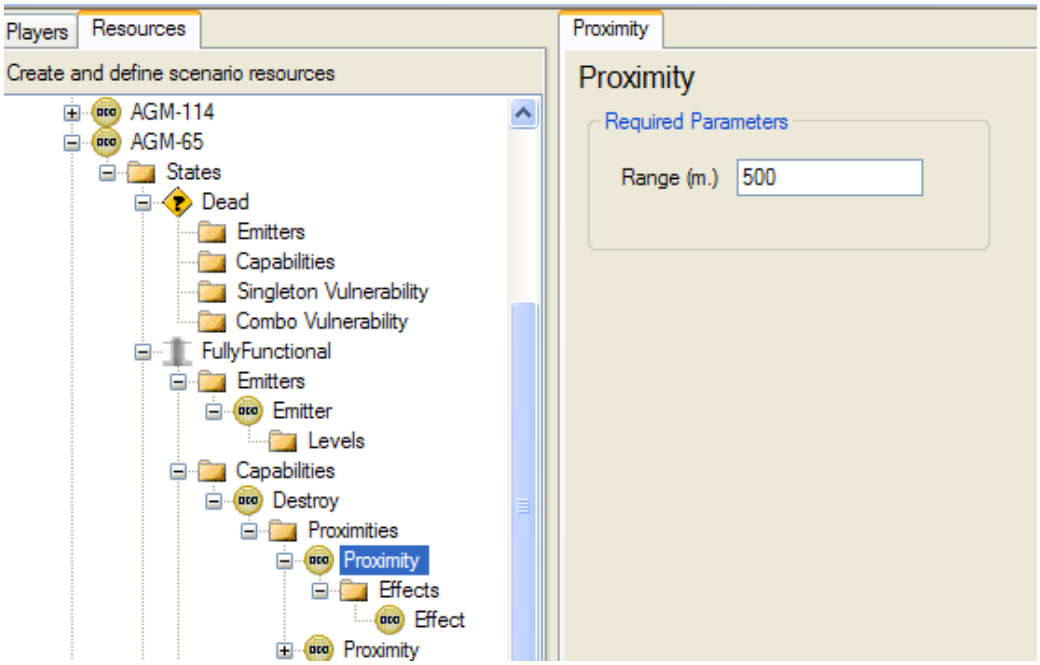


When a sensor detects only a single attribute, then its range can be specified as a Sensor Range. When a single Sensor Range has been defined for a sensor, the range might be limited to a specific value (see below). As above, a value of zero indicates that the range is unlimited.

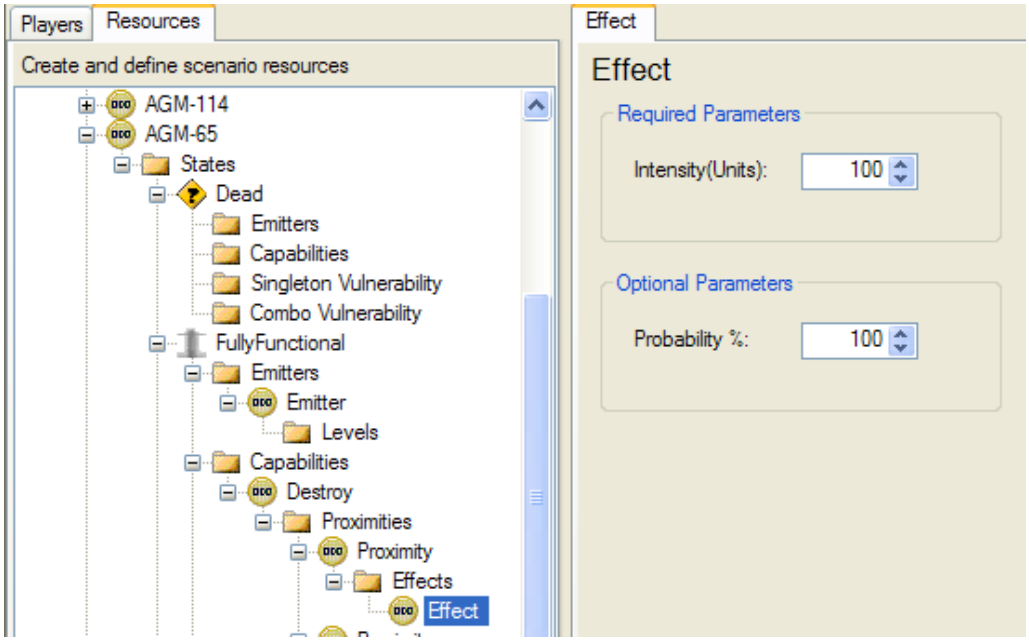


How Can Attack Abilities be Made to Depend on Distance?

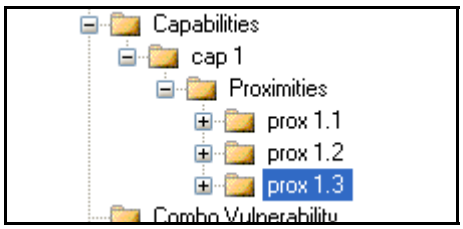
Attack capabilities ("Capabilities") always depend on distance. In the simplest case, the first step is to name the capability and its range of operation (as shown below).



Then indicate the strength of its attack ("Intensity") as an effect.



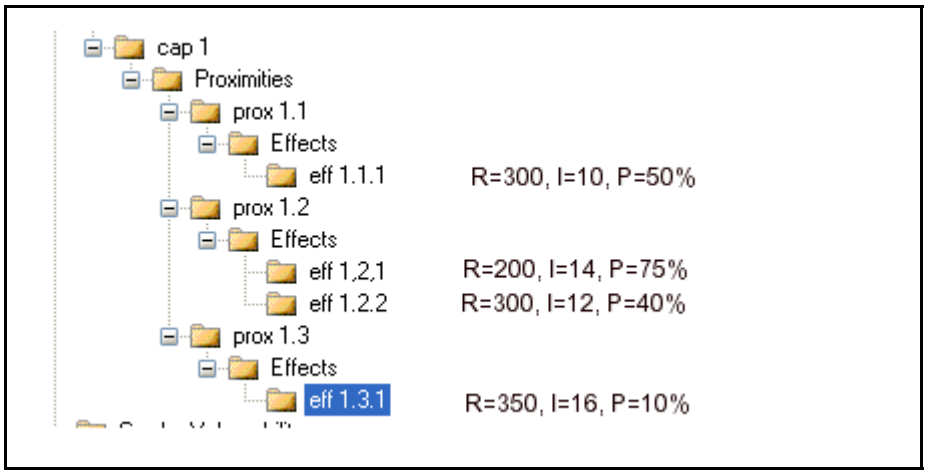
However, for more complex interactions, the scenario writer can define the capability to have different effects at different ranges. Typically, each effect would have a different range, intensity, as possibly probability of success, with closer proximity correlated with greater intensity. The next figure illustrates a capability with three proximities.



How Can Uncertainty be Built into the Action of a Weapon?

The previous example shows that an effect has a probability. Giving an effect a probability of 60% would mean that the weapon was successful 60% of the time within the range associated with that proximity element.

However, even at a given range there might be different effect clusters. For example, consider the capability shown below.



At 300 meters, there is a 50% chance that the attack will succeed with an intensity of 10, and a 40% chance that it will succeed with the larger intensity of 12. In a situation like this, the various effects are tried in the order they are given in the scenario.

How Can the Scenario be Made to Require Cooperation for an Attack to be Successful?

The simplest way to do this is to define a vulnerability in the target that cannot be met by any single unit. For example, if a unit is vulnerable to 80 units of intensity from some capability, but each of the potential attackers could only bring 50 units to bear, then two attackers would be required.

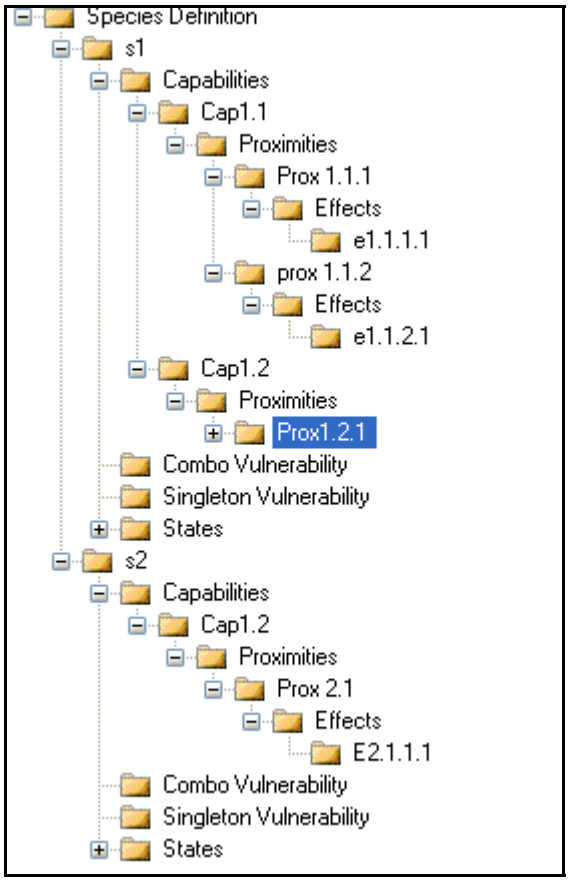
In this situation, it could also be that a single unit could bring the required intensity to bear by quickly firing twice. This can be enabled or disabled by setting a value for Default Engagement Time, which appears on the Scenario Definition screen. The engagement time begins at the first attack. During that period, all attacks are taken together. So setting the engagement time at two or three seconds would be enough in practice to prevent one unit from firing twice as a part of the same engagement.

It is also possible to create more complex scenarios in which different capabilities must be combined for an attack to succeed.

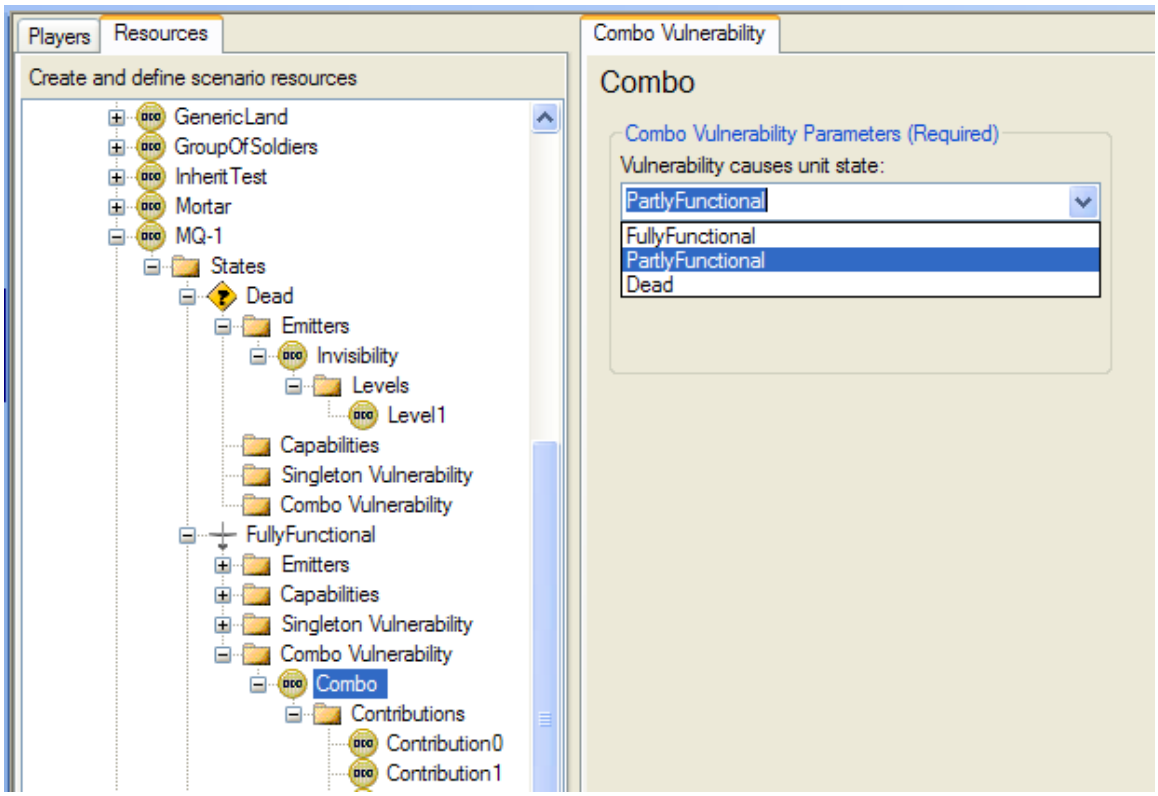
A Combo Vulnerability ("combo") is a list of capabilities (as illustrated in the table below), all of which must be successfully applied to a unit in order for it to be affected.

Species	Capability	Range	Intensity
s1	Cap1.1	100	50
s1	Cap1.1	50	80
s1	Cap1.2	100	50
s2	Cap1.1	50	80

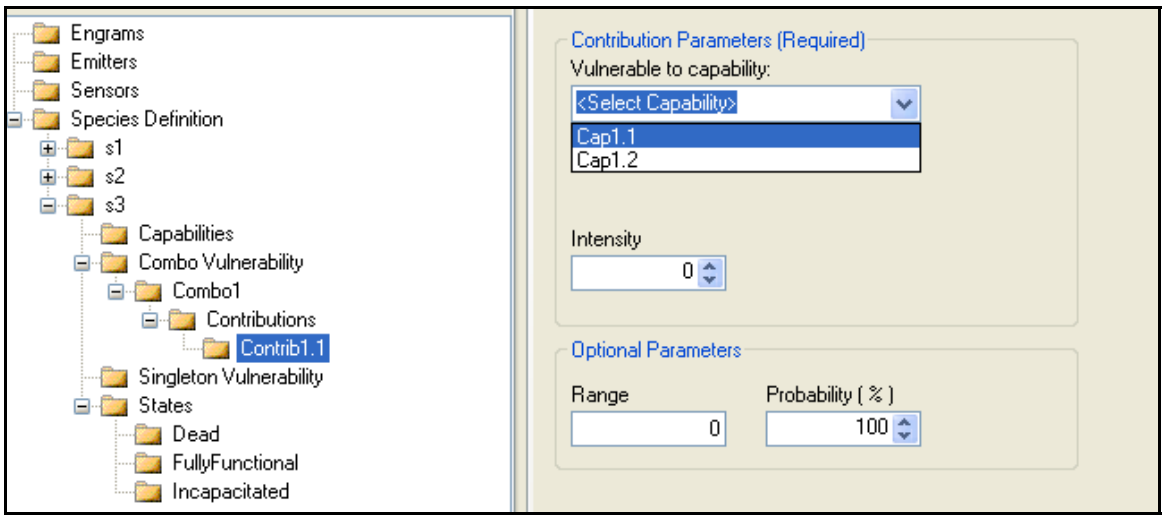
The next figure shows two species, with the following capabilities.



Suppose the scenario writer wants to create a species called s3 that could be affected only if units from these two species work together. This can be accomplished by defining one or more combos for s3. A combo is defined by a collection of Contributions. Each Contribution is a single requirement about the attack on the unit–all of the requirements must be satisfied for the state transition to take place.



A Contribution specifies a capability, an intensity, and (optionally) a range and probability.



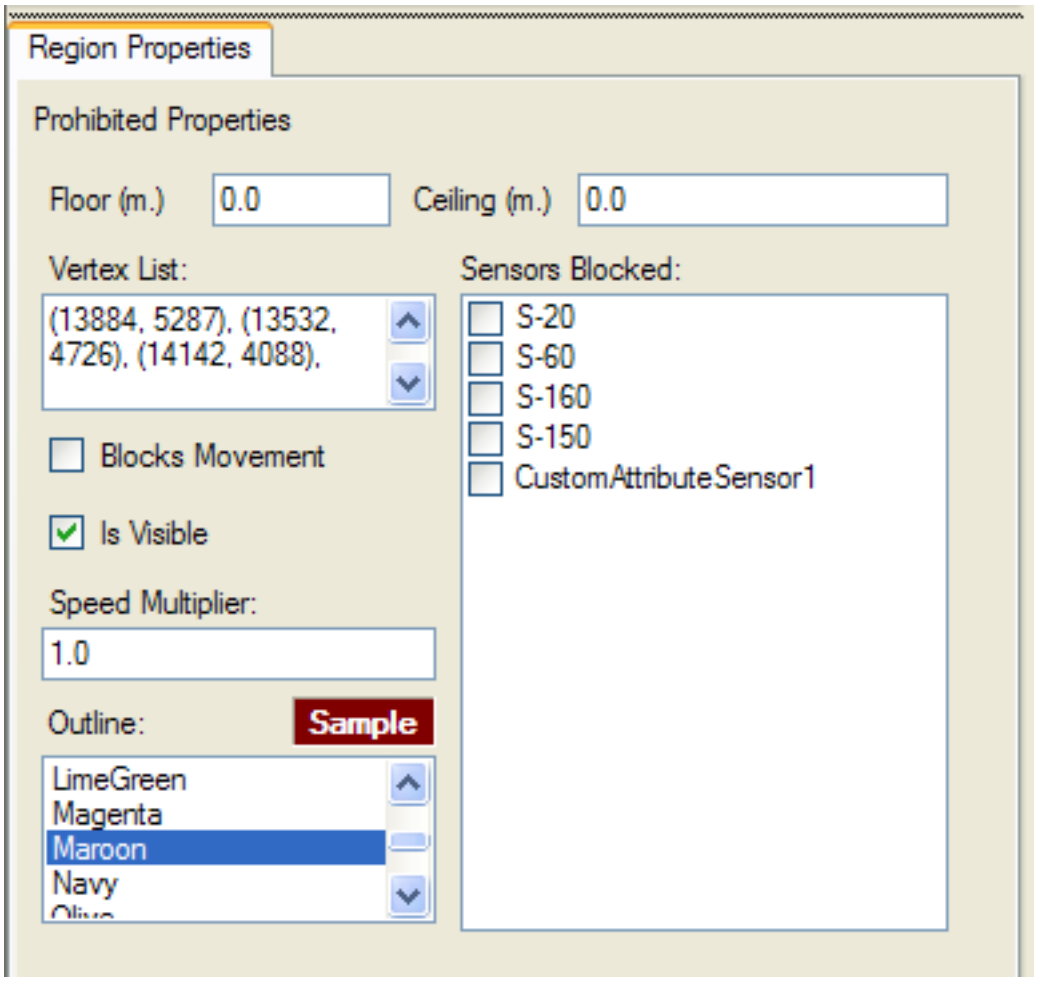
Suppose there is a Combo for species s3 that has two contributions, described in the table below.

	Capability	Intensity	Range	Probability
Contribution 1	Cap1.1	60	-	100
Contribution 2	Cap1.2	45	-	100

Then for the state change associated with that Combo to take place, the unit must be attacked by Cap1.1 with an intensity of at least 60, and by Cap1.2 with an intensity of at least 45. Checking the table above, notice that this can be done if species s1 (meaning, the unit belonging to this species) is within 50 meters and uses both capabilities. But if s1 is within 100 meters, then the attack requires s2 to also apply Cap1.1–each can only bring an intensity of 50 to bear at that range, but the combined intensity is 100. That is, by getting close enough and deploying both capabilities within the engagement time, a single unit from species s1 can attack successfully. However, at a greater distance it needs an assist by a unit from s2. Arbitrarily, many different contributions can be defined for one Combo–they are applied in the order they are listed.

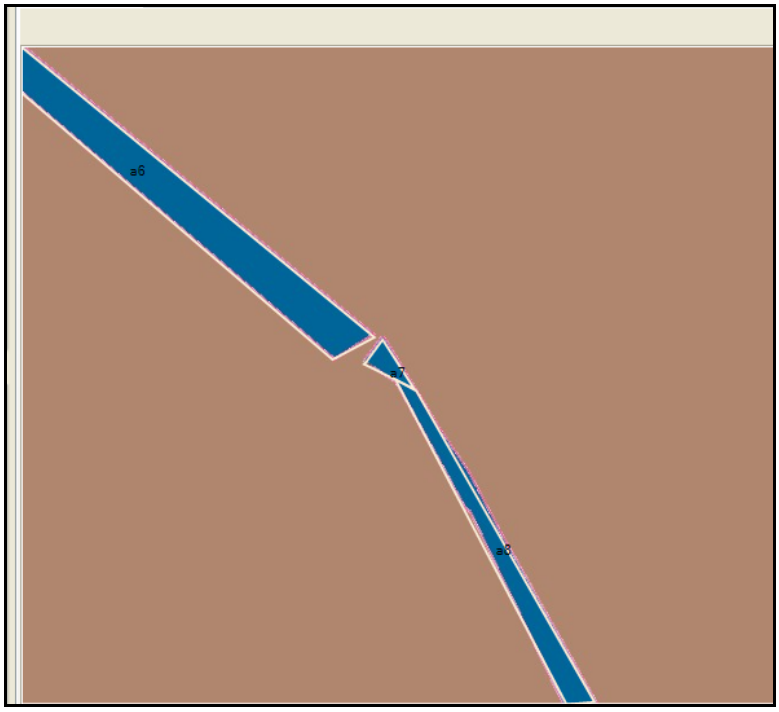
How Can Different Terrains have Different Travel Characteristics?

Active regions can block or slow down movement. They can also block signals. These characteristics are set when the region is defined, as shown below.



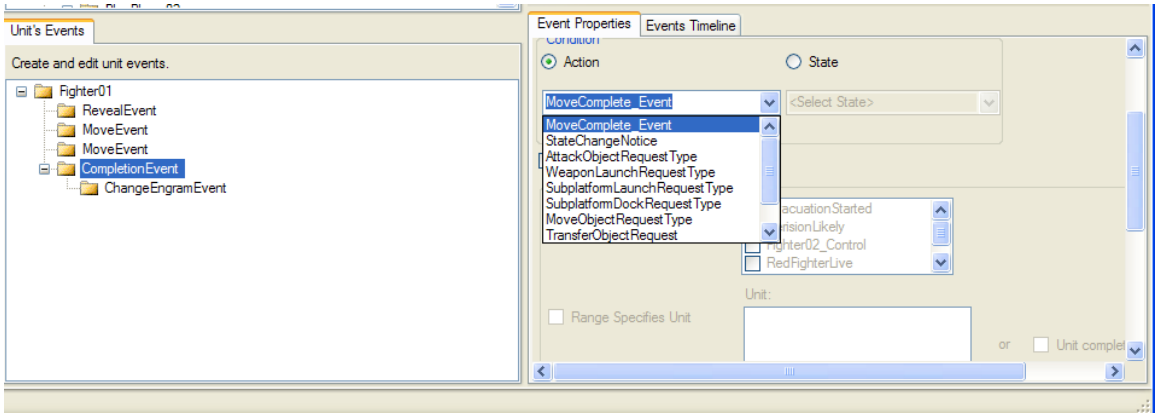
How Can Units be Restricted to Travel in Certain Areas?

Units can be restricted to travel in certain areas by blocking travel outside of them. For example, in the following map, to make the blue areas impassible, cover them with active regions that had the Blocks Movement feature selected (shown with white outlines).



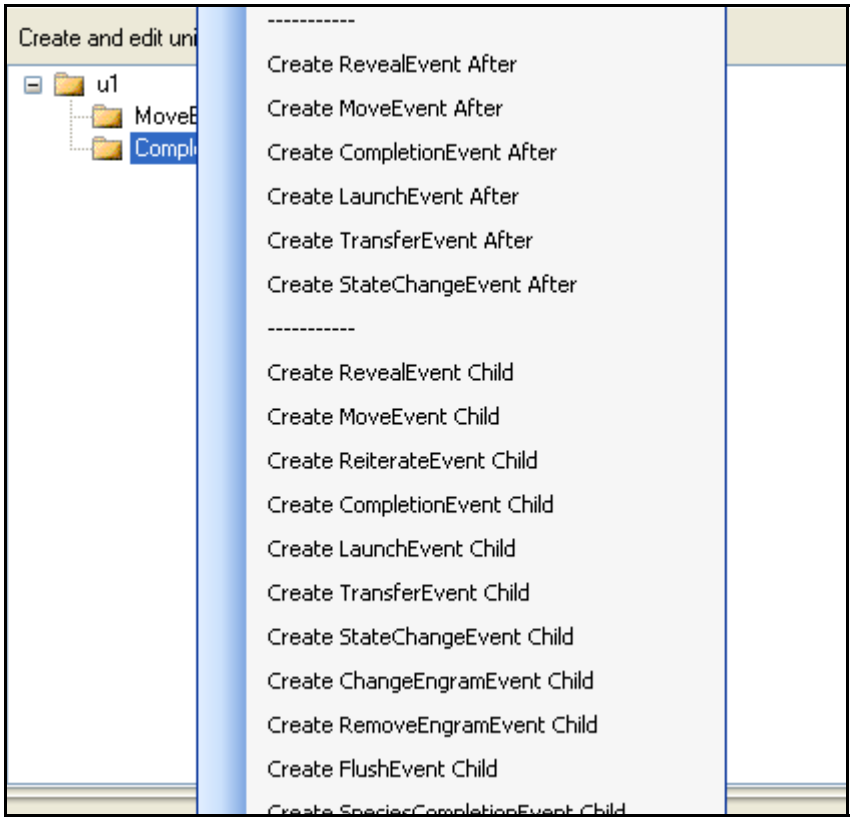
How Can Events be Made to Depend on Whether Other Events Have Occurred?

There are many ways to arrange this kind of linkage. They all begin with the notion of a Completion Event, which is an event that only happens as a result of some other event. This figure shows a dialog from which the kind of event being monitored can be chosen.

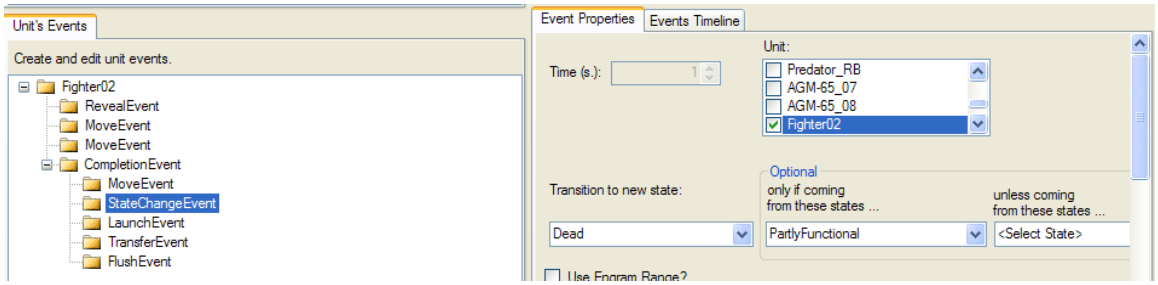


Suppose that "MoveComplete_Event" is selected. The meaning is that some other event is to occur directly after the movement of some unit is completed (in this case,Fighter01 as shown on the left).

The dialog that selects which event is to occur is shown below.



Suppose that "Create StateChangeEvent child" is selected, as below.



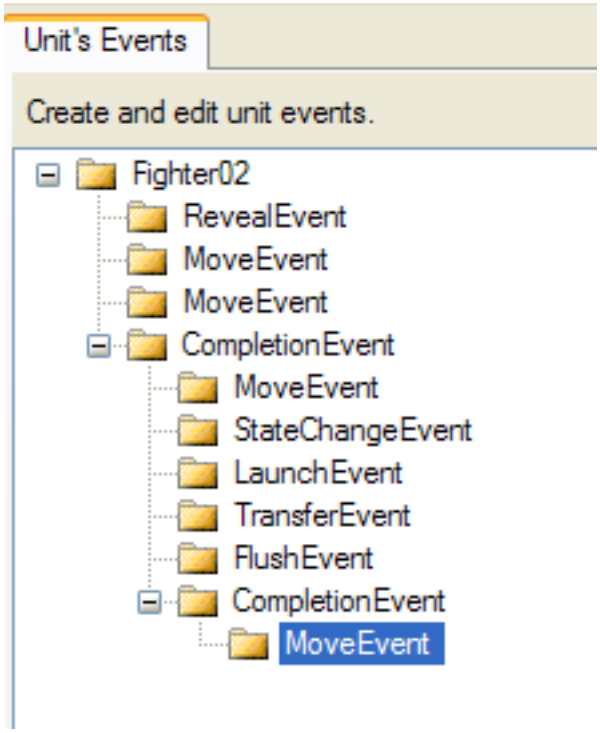
It is now possible to specify that when unit Fighter02 completes a move by reaching the intended destination, some unit will change state.

The contingent event happens immediately after the Move Event is complete. Once a Completion Event is used, it is no longer available. Therefore, to create a sequence of actions it is necessary to create a chain of Completion Events–this is essentially the function of the Reiterate command, although that is limited to responding to completions of moves.

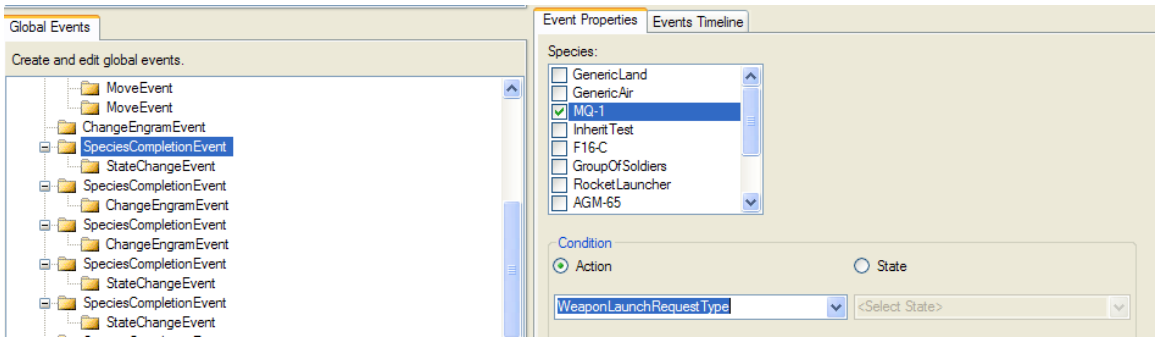
Incidentally, the Completion Event is not tied to a specific Move Event. That is, suppose a unit is made to move toward location A but that halfway along it is redirected to location B. Upon reaching B, the very next Completion Event that specifies a MoveComplete Event for that unit will be triggered.

Since the Completion Event is only used once, it is not possible to use this mechanism to say that, for example, every time that unit Fighter02 completes a motion some specific event happens

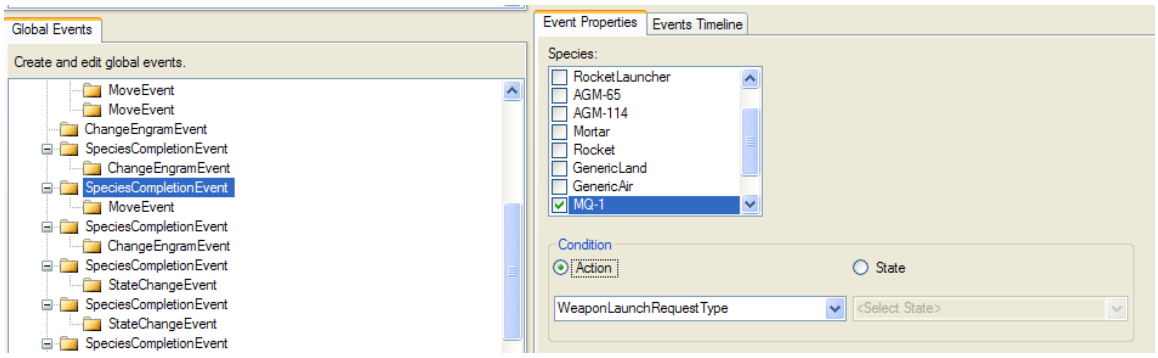
(such as putting some other unit into a particular state in case it has drifted out of it). The best that can be done is to make a long sequence, as shown below.



There is another kind of completion mechanism, however, that can cause the same response to occur on every completed Move Event (or other type of event). This is the SpeciesCompletion Event. A SpeciesCompletion Event is similar to the Completion Events just discussed, but there are two significant differences. Where the Completion Event is tied to a specific unit, the SpeciesCompletion Event is triggered when an event involving any unit of a given species occurs. In addition, the SpeciesCompletion Event is not removed when it is triggered. Therefore, it can come into play repeatedly. The next illustration shows the process of defining a SpeciesCompletion Event that will be triggered when any unit of species MQ-1 launches a weapon.

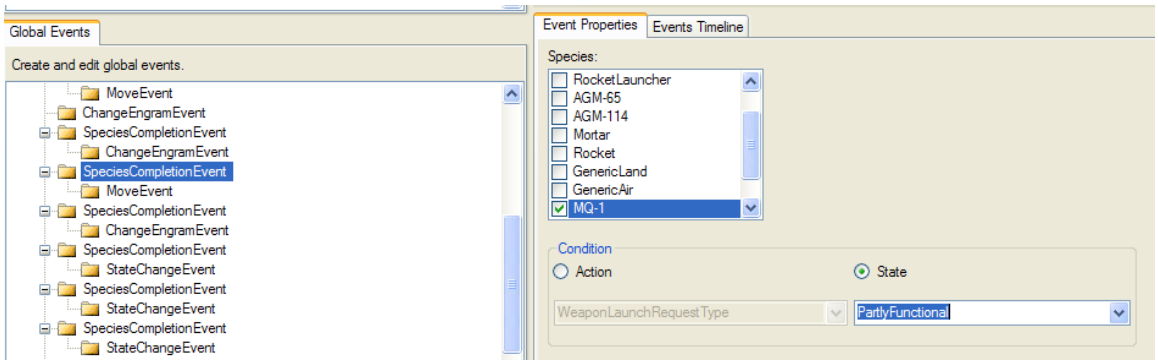


When a SpeciesCompletion Event is triggered, the resulting action can involve any unit, as is true for the Completion Event. However, there is an additional option, which is to specify that the resulting action involve whatever unit it was that triggered the SpeciesCompletion Event. Continuing the previous example, the next figure shows how to specify that whenever a unit from species MQ-1 launches a weapon, that unit should then immediately move to a specified location.



How Can I Make One Unit Act as a Result of Doing Something to Another?

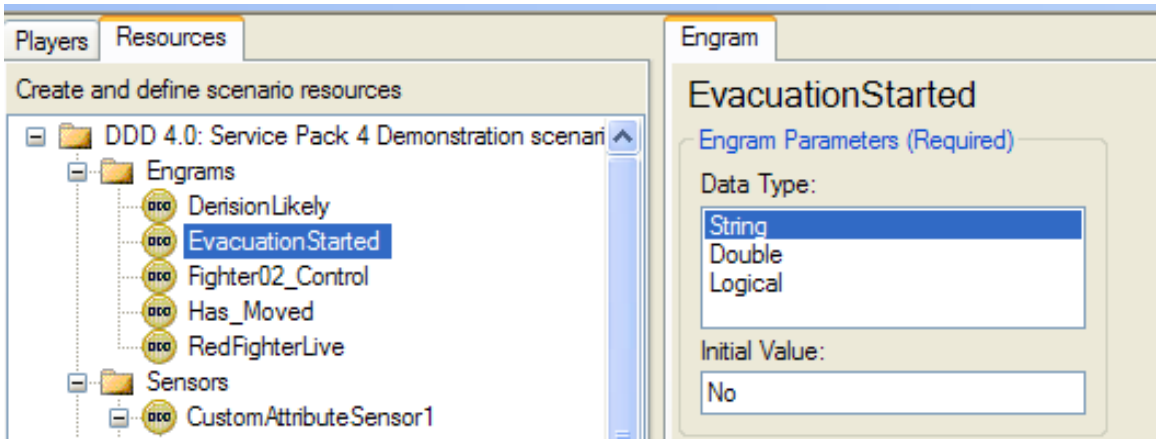
This also involves the Completion or SpeciesCompletion Events. However, in this case, use the State option, which causes the event to trigger whenever there is a state change. For example, suppose the scenario writer wants to cause a unit to return to its base if it is only partially functioning but not destroyed. To do so, give the unit a state called PartlyFunctional, which it will be put into if some of its vulnerabilities are breached, and create a Completion Event to watch for a change to state to PartlyFunctional. If that Completion Event is triggered, the unit will move to a predefined location, as shown below.



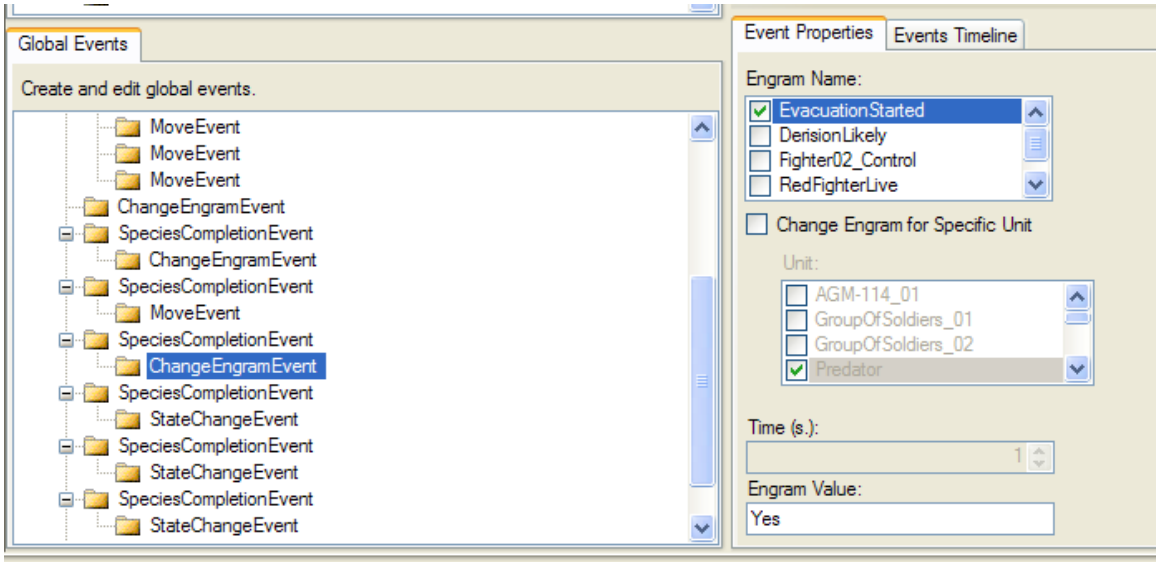
How Can I Make the Course of the Scenario Depend on Events That May Have Happened Arbitrarily Far Back or May Have Been Associated with any of a Number of Units?

Completion Events and SpeciesCompletion Events specify reactions but do not provide any long-term memory. Memories can be created or referred to with Engrams. An Engram is a string whose value can be set or tested in a scenario. For example, suppose the scenario writer wants the course of a scenario to depend on whether an evactuation has started.

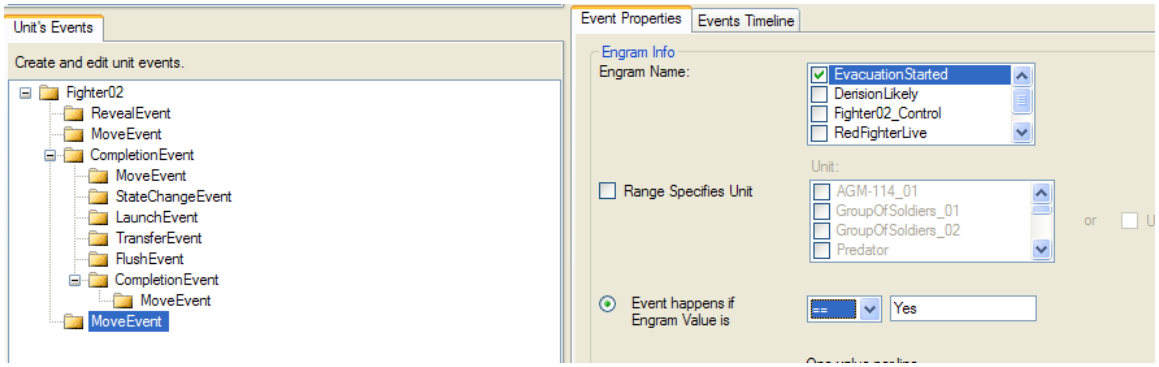
To accomplish this task, begin by defining an Engram, as shown below.



Next, a SpeciesCompletion Event is needed to detect whether an evactuation has started.



Other events can be made to depend on the value of this engram by building an Engram Range condition into the event definition. The next illustration shows how to set up Fighter02 to move only if an evactuation has started (i.e., EvacuationStarted value is Yes).



Not seen in the above illustration is the option of specifying values of the engram that will prevent the event from happening. Thus, the same effect could have been achieved by leaving the "Event happens if engram is one of these values" box empty, but filling in the "Event happens if engram is NOT one of these values" box with "true."

How Can I Make Two Planes Take Turns Flying Along Some Route?

This situation can be easily accomplished with a Reiterate Event. Suppose the scenario writer wants two units, A and B, to move in alternation by using a Reiterate Event.

The underlying principle is that when each Move Event is started, a Completion Event for it is issued to watch for its completion. When the Move Event (for A) completes, the next Move Event (for B) is started and the corresponding Completion Event is set up.

Exactly the same approach can be used to choreograph any collection of units.

So long as none of the units is interfered with, the sequence will continue. However, if any of the units fails to complete its assigned move, the entire sequence comes to a halt. Conversely, the sequence is not easily interfered with, as long as the moves (or some moves) complete. Thus, for example, if in the above example unit A were to be caused to move someplace else by a live player, the original MoveComplete notice would not be generated and unit B would not do anything. However, when the redirected Move Event completes, unit B would begin its trip, and when that was complete Unit A would start moving to its next waypoint.