HOMEWORLD2

GAME RULE MODS

Introduction

This document describes how to make Game Rule Mods for the HW2 engine. Game Rule Mods are mods that define rules for gameplay in multiplayer games. They are not to be confused with command-line mods, which are described in a separate document.

This document is a companion to the example Game Rule Mod, called Resource Race, which is available from http://www.relic.com/rdn/ This is a very simple game type. The game is won by the first person to reach a certain target amount of resources. With this mod, you will have an example of how to:

- Name and describe your mod
- Specify a directory for your game-type-specific levels
- Specify options specific to your game type
- Create game rules to determine the winner.
- Display special on-screen UI.

Prerequisites

In order to create a game-type mod, you will need the following tools and documentation:

All files are available at http://www.relic.com/rdn/

Archive.exe – the tool which is used to create .big files. This is how game type mods are distributed.

SCAR documentation – SCAR is the language HW2 AI scripts are written in. It is Lua with HW2-specific extensions.

Uncompressed ResourceRace – The contents of ResourceRace.big and the script needed to create it.

General implementation

Game Type Mods are created as a bigfile, or archive (with the extension .big) and stored in the bin\GAMERULES directory. At startup, HW2 scans this directory and loads basic description info from any bigfiles it finds. When a Multiplayer or Skirmish vs. CPU game is created, the user can choose from available game types. The basic game rule, Deathmatch, is always present. Any game rules found on the local machine are also added to the list of available game rules.

In the multiplayer lobby, both GameSpy and LAN, the name of the game rule is listed. If a client does not have the correct .big file on their computer, they cannot join the game.

When a game is finally started, the .big file is loaded. Any files in the .big file that exist in the main game bigfiles (Homeworld2.big etc.) will be overridden by the data files existing in the Game Rules Mod.

Bigfile format

Bigfiles are a way of packing a bunch of files into one larger file. This makes loading faster and makes distribution easier. Within a bigfile, there are many files that are arranged into different Tables of Content, or TOCs. Game Rule Mods need at least one TOC: data. Within the data TOC, there needs to be at least one file: LevelData\Multiplayer\<GameRules>.lua, where <GameRules> is the same as the name of the Game Rule. This file defines the multiplayer game rules for the mod, specifies options and more.

Also within the mod can be any number of level files, localization data for one or more languages as well as generic data such as new ships.

<GameRules>.lua

To define a new gametype, you need to create a <GameRules>.lua and store it in Data/LevelData/Multiplayer. This can reside in either a mod or on the local hard drive, but it needs to be in /LevelData/Multiplayer. At startup, and MD5 checksum of this file is made to uniquely identify it. That way, if there are multiple versions floating around, only identical game rules can be used against each other.

<GameRules>.lua has the following format:

Name	Type	Description
GameRulesName	String or quoted locID	Localized name of the game rules. Can be a string or a localization id such as “$8000”
Description	String or quoted locID	Localized description of the game rules. Gets displayed in the game setup screen. Limited to 256 characters.
Directories	Table of named strings	A table of named directory paths. Currently only Levels = is supported. If specified, it is a directory where levels specific to the game type may be stored.
GameSetupOptions	Table of tables	A table of game setup options. Refer to following table for specific formats.

OnInit	Function	Called when the script starts. Add at least one watch rule here to execute each frame.

The entries in the "GameSetupOptions" table have the following format:

Name	Type	Description
Name	string	The name of the choice that the code and scripts use.
locName	String or quoted locID	Localized name as displayed in the game setup screens.
Tooltip	String or quoted locID	Short description of the option for the tooltip.
default	int	Default option as an index into the choices table.
Visible	bool	0 means the option will not be visible to the user, but will be visible to code. Option will always take default option. This is useful for options that the mod does not want adjusted.
choices	table of locID/string pairs	An array of the options arranged as locID/string pairs. The locID becomes a localized string that the user sees. The string is the name of the option that the code and scripts use.

The rest of the file (including the OnInit function) is regular SCAR code. Please refer to SCAR documentation for details on the functions available.

Localization within mods

For Game Rules Mods, you can add localization content to mods by creating an archive with multiple TOCs. If one of the TOCs has the alias "locale" it will modify locale data. You can put several languages into a single mod by defining multiple TOCs all with "locale" as the alias, and the language in the TOC name. Note that the name of the TOC has to be unique among all loaded archives, so you should make sure that you include the language name AND the mod name/version in the TOC name.

With multiple locale TOCs in a single archive, only the one that matches the current language, or "English" if current language not found, will be mapped into the "Locale:" filesystem. This allows mods with multiple languages to play against each other in multiplayer games. Players playing with different languages and seeing different text on-screen will not go out of sync.

Refer to the script included in the uncompressed Resource Race data for an example of how to create archives with multiple TOCs.

Please note that the localization ID range of 8000-8999, inclusive, is reserved for Game Rules Mods. Therefore, you should keep your localization IDs within this range to prevent problems.

Creating game rules

Within the OnInit() function in <GameRules>.lua, you should do any required initialization and create at least one watch function. A watch function will be called each simulation step and this is how you will determine the game end condition. You can add and remove additional rules as the game progresses if you like.

Displaying special on-screen UI

You can display special UI from your <GameRules>.lua by calling the ATI_xxx() functions from your watch functions. This is done by calling the ATI (Advanced Tactical Interface) SCAR

functions. Please refer to the SCAR documentation for more info on these functions. Refer to the Resource Race example for an example of it's usage.

Briefly, you must load a file that describes one or more ATI Templates that contains a number of items to display. Your script can then display these templates at a specified 2D location on-screen, with any number of optional parameters.