D2Moddin Documentation

Release 1.0

Christian Stewart

August 10, 2014

Contents

D2Moddin is a platform to easily play Dota 2 custom game modes. It is also a base for the Dota 2 modding community, with the D2Moddin forums and the developer documentation you are reading right now.

CHAPTER 1

SDK Setup

Dota 2 game modes are built using the addon framework used to build the official Frostivus and New Bloom events. Addons are essentially overlays on the game directory - any content in an addon will override the default content. Game code is written in lua and operates server-side only. Custom assets (models, sounds, and UI) can be added to the client and used in maps.

As there is currently no officially released toolchain, including map and particle editors, the Alien Swarm SDK and pipeline is used to create maps and assets.

See the below documentation pages to get started:

1.1 SDK Setup Tutorial

This page is based on the Dota 2 wiki.

Install the Alien Swarm SDK. In Steam, go to the Library section, and under Tools, download the Alien Swarm SDK. You can also click this shortcut link.

1.1.1 Extract Dota's Files

Dota 2's content files are distributed in compressed VPK form. You can extract them using GCFScape to later modify them in your game mode. You may need to use the x86 version, which works fine on 64 bit systems.

The primary Dota 2 package is the `pak01_dir.vpk` file under the base dota game directory. Open the pak in GCFScape, right click the root element, and extract the files to `Alien Swarm\swarm\addons` so they may be used in the Alien Swarm editors.

Rename the directory you have just created in the Alien Swarm addons to `Dota2Extract`.

1.1.2 Misc. Files

Copy the `bin` and `platform` directories into the `Dota2Extract` directory.

Finally, download dota2fgd (by RoyAwesome on GitHub) and UpVersion.exe (by ModDota) to the bin directory within `Dota2Extract`.

1.1.3 Alien Swarm Search Paths

Replace the contents of `Alien Swarm\swarm\gameinfo.txt` with:

```
"GameInfo"
{
        "game"    "Alien Swarm"
        "title"   "    "  // asw - leave this blank as we have a texture logo
        "type"    "multiplayer_only"
        GameData  "swarm.fgd"
        InstancePath   "tilegen/instances/"

        SupportsDX8     0

        "FileSystem"
        {
                "SteamAppId"  "630"
                "ToolsAppId"  "211"

                "SearchPaths"
                {
                        "Game"  "|gameinfo_path|."
                        "Game"  "swarm_base"
                        "Game"  "platform"
                        "Game"  "|gameinfo_path|addons\Dota2Extract"
                }
        }
}
```

1.1.4 Configuring Hammer

Start the Alien Swarm SDK, select Hammer World Editor and let it open up. Now go to "Tools" -> "Options" and add the "dota2.fgd" from `Alien Swarm\Swarm\addons\Dota2Extract\bin\dota2.fgd`.

Next, increase the render distance so that the entire map will be visible at any given time. Under "3D Views" increase the Model Render Distance and Detail Render Distance.

1.1.5 Creating a Test Map

Under File, create a new map. Save it as `test.vmf`. Next, select File->Run Map and press "Expert" in the bottom left corner:

Click "Edit" and create a new config for Dota 2. We will define the build process as a series of commands.

The BSP Command

Click "New", then "Cmds" and "BSP Program". Add `-alldetail -game $gamedir $path\$file`.

UpVersion

Upversion is a tool that will convert the BSP into a format suitable for Dota 2.

Click "new" then "cmds". Select "Executable" and find "UpVersion.exe" in `Alien Swarm\swarm\addons\Dota2Extract\bin`. If it's not there, return t the Misc Files step and download it.

Add the parameters `$path\$file.bsp $path\$file.$ext $path\$file.pbm`.

Copy Map

Finally, to use the map in Dota 2 we will copy the compiled .bsp to the Dota 2 maps directory.

Click "new" then "cmds". Select "Executable" and specify: `CgWindows\System32\xcopy.exe`, the built in Windows copy tool. Add the parameters: `$path\$file.bsp "C:\Program Files (x86)\Steam\SteamApps\common\dota 2 beta\dota\maps" /y`. Make sure the path is the correct path to your Dota 2 maps directory.'

1.1.6 Next Steps

You now have the Alien Swarm SDK set up properly to start working on Dota 2 game modes. You can follow the mapping tutorial in the next section.

1.2 Creating an Addon

"Addon" and "Game Mode" are used interchangeably throughout this documentation.

1.2.1 Addon Bootstrapper

The D2Moddin addon bootstrapper will ask you some simple questions about your game mode and build the files for your mode based on your answers. In the future it will also be able to create additional files for you for NPCs, units, particles, maps, etc.

To use the bootstrapper, you will need Node.JS. Download and install it.

First, download the latest version of the bootstrapper. You can use `git clone --recursive https://github.com/D2Modding/d2tool.git` or just download from here.

If you downloaded from the link you need to also download barebones and put it into the `barebones` directory in d2tool. Otherwise you will see an error saying `info.json` is missing.

Use the `cd` command to move into the d2tool directory.

Install the dependencies with NPM, run `npm install` in the d2tool directory.

Next, run the bootstrapper with Node.JS. `node d2tool.js`. In the future there will be a compiled exe for this as well.

Answer the questions:

The result will be the following directory structure (in this example, a mode called "teamfight"):

```
-- addoninfo.txt     - Version & name information.
-- info.json         - D2Moddin metadata
-- maps              - Maps
|   -- teamfight.bsp - Compiled map
|   -- teamfight.gnv - Compiled gridnav
-- materials         - Any engine materials
|   -- overviews     - Minimaps (named Overviews)
|       -- teamfight.vmt
|       -- teamfight.vtf
-- particles         - Custom particles
```

```
|   -- frostivus_gameplay.pcf
|   -- frostivus_herofx.pcf
|   -- test_particle.pcf
-- PhysicsReadme.txt - Readme of the physics system
-- resource          - Translations & flash resources
|   -- addon_english.txt
|   -- flash3
|   |   -- images
|   |       -- items
|   |       |   -- example_item.png
|   |       -- spellicons
|   |           -- holdout_battle_rage.png
|   |           -- holdout_blade_fury.png
|   |           -- holdout_culling_blade.png
|   |           -- holdout_fiery_soul.png
|   |           -- holdout_focusfire.png
|   |           -- holdout_friendly_skewer.png
|   |           -- holdout_glacier_arrows.png
|   |           -- holdout_gods_strength.png
|   |           -- holdout_guardian_angel.png
|   |           -- holdout_multishot.png
|   |           -- holdout_omnislash.png
|   |           -- holdout_scourge_ward.png
|   |           -- holdout_voodoo.png
|   |           -- templar_assassin_refraction_holdout.png
|   -- overviews - Define any custom overviews
|       -- teamfight.txt
-- scripts       - Scripts and other game mode data.
|   -- custom_events.txt
|   -- game_sounds_custom.txt
|   -- maps
|   |   -- barebones.txt
|   -- npc       - KV files for heros, items, units
|   |   -- herolist.txt
|   |   -- npc_abilities_custom.txt
|   |   -- npc_abilities_override.txt
|   |   -- npc_heroes_custom.txt
|   |   -- npc_items_custom.txt
|   |   -- npc_units_custom.txt
|   -- shops     - Shop data
|   |   -- barebones_shops.txt
|   -- vscripts  - Server-side scripts (can be excluded in the client)
|   |   -- addon_game_mode.lua
|   |   -- addon_init.lua
|   |   -- physics.lua
|   |   -- teamfight.lua
|   |   -- util.lua
|   -- world_map_custom.txt
-- sound        - Custom sounds
    -- mini_rosh_firebreath.wav
```

CHAPTER 2

Mapping

Mapping is done in the old Source Engine map editor, Hammer.

See the below documentation pages to get started:

2.1 Mapping with Hammer

Maps are created with Hammer for Dota 2, an old Source Engine map creation tool used for Portal 2, Half Life, and other valve games.

Make sure you've completed the getting started tutorial first.

2.1.1 General Mapping Notes

- Maps are a much larger scale in Dota 2 than they are in other Source games. Units are around 48 units wide, towers are 128x128x320 units tall.
- If an entity is outside the bounds of the map, it crashes the game with no error.
- AlienSwarm converts all brushwork into func_detail, so you need to seal the map with a func_brush named structure_seal. The brush should be textured nodraw.
- Required entities include: info_player_start_goodguys, info_player_start_badguys, ent_dota_game_events, env_global_light.
- Models require a shader that is not provided in Hammer. In your extracted gamedata, open every .vmt file and replace "GlobalLitSimple" with "VertexLitGeneric". A tool such as Notepad++ can do this quickly.
- The func_brush named "structure_seal" should be nodraw (no skybox required).

2.1.2 Overlay Limit

You cannot decompile and re-compile the dota.bsp map as it has too many overlays. The overlay limit was raised to 8192 from 512 for Dota 2, but the compiler toolchain still works under the 512 limit.

2.1.3 Fog of War Calculation

Units can climb to virtually any height within the map, but there are only five Fog or War heights defined in the Dota 2 engine. They are:

- 0-128: River
- 128-256: Midlane
- 256-384: Highground base
- 384-512: Ward spots
- 512-∞: Edge of the map.

The camera will stay in place above 512 units in the Z axis.

2.1.4 Building the NavGrid

Units in Dota 2 do not navigate based on the structure of the map, rather, they move on a 2D grid known as the navmesh. Every square in the navgrid has two states: pathable, and not pathable. If a square is not pathable, the unit will not walk there and cannot blink/force staff to that location. It is possible to cross unpathable areas with blinks or force movements.

Install the GNVTool

The GNVtool (by Penguinwizzard) converts netpbm formats to the binary GridNav format. It can do two way conversion between PBM files, which are editable in photo editing software to manually touch up the grid.

- Download GNVTool
- List of NetPBM Software
- Paint.Net PBM Plugin

Netpbms are 1 bit bitmaps. The bitmap size must be the world size / 64. The bitmap must have dimensions divisible by 8 or it will be sheared.

Usage: code::`GNVtool.exe tognv source.pbm target.gnv offsetx offsety`.

Offset x/Offset y are always negative and 1/2 the dimensions of the bitmap.

You can view your navmesh in game with code::`dota_gridnav_show 1`.

2.1.5 Creating a NETPBM

First, open up a Hammer map.

The first two arguments you need are the dimensions of the map. This will be a rectangle aligned with the Hammer grid.

In this image it is 4096x4096:

Next, you need the offset of the map from the origin. Get the top left point of the map (in this image, -2048, 2048).

Using these values, you can create the netpbm. Divide the dimensions by 64 to get the size of the pbm file you need to make (here 64x64). Use the offset (with the y-axis-component negated, because gnv files are upside-down) to calculate the last 2 arguments to GNVTool, and you should get a gnv file that works for your map!

2.1.6 Creating a Minimap

The VTFEdit tool is used to convert an image into a .vtf minimap file.

While Valve paints their minimaps manually, it is generally sufficient to take a top-down screenshot of the map.

Open VTFEdit, select Import, find your image, and save it as MAPNAME.vtf. Under tools, select "Create VMT". In the Options tab, for shader, select UnlitGeneric and check the boxes 'Translucent' and 'Vertex Alpha'. Save it as MAPNAME.vmt.

Put both of these files in a folder named materials/overviews in your addon directory.

Next, create a new textfile named MAPNAME.txt. This is a Key-Value file denoting where the bounds of the minimap lie. The structure is as follows:

```
MAPNAME
{
material "overviews/MAPNAME" //Note no file extension
  //Coordinates to the upper left corner of your map
  pos_y 2560
  pos_x -2560
  scale 5.000 //Minimap scale.
  rotate 0 //Minimap rotation.  This should always be 0.
  zoom 1.0000 //Minimap zoom.  This should always be 1 unless your texture is larger than the playabl
}
```

Put this file in your code::`addon/resource/overviews` directory.

CHAPTER 3

Game Logic

Game logic is written in LUA.

See the below documentation pages to get started:

3.1 LUA Scripting Guide

This is a short documentation on the various parts of LUA scripting in Dota 2. You can refer to the API documentation for specific functions/game events etc.

3.1.1 Addon Initialization

When the server starts up and loads your game mode, it will first execute `addon_init.lua`. In the bootstrapped addon, `addon_init.lua` will require util, physics, and finally your game mode lua file at the end of the script.

Next, when the server is ready to create your game mode instance, it will execute `addon_game_mode.lua`, which, in the bootstrapper, will simply call `InitGameMode()` on your game mode defined in `gamemode.lua`.

3.1.2 Lua Scope

Any variables defined in the root of the script, without the `local` tag, will be considered global and accessible anywhere in any lua file loaded into the Dota 2 Lua VM.

You will generally want to use the `local` tag before variables so you don't pollute the global scope.

3.1.3 Game Mode Class

Game modes are Lua objects/tables, defined as such:

```
TeamFightGameMode = {}
TeamFightGameMode.szEntityGameMode = "gamemode"
TeamFightGameMode.szNativeClassName = "dota_base_game_mode"
TeamFightGameMode.__index = TeamFightGameMode
```

This definition block occurs in `gamemode.lua`, which will be named according to your mod's name.

Next, the functions that the mode requires are defined on the game mode table/object, for example:

```
function TeamFightGameMode:InitGameMode()
  ...
end
```

3.1.4 Registering Hooks

Most of the logic in the lua game mode code revolves around hooking into the game's standard events.

For example, to perform some logic when a player says something, register the hook in the game mode init:

```
ListenToGameEvent('player_say', Dynamic_Wrap(TeamFightGameMode, 'PlayerSay'), self)
```

Here, we ask the Lua engine to call `TeamFightGameMode:PlayerSay(keys)` when the `player_say` game event is fired.

Next, define your implementation:

```
function TeamFightGameMode:PlayerSay(keys)
  local ply = self.vUserIds[keys.userid]
  Log(keys.text)
end
```

In this case, `keys` is a table/object with relevant data to the say event, such as the text of the message.

You can view the full API in the API docs sections.

3.1.5 Timers

To put off execution of some code into some seconds in the future, you can register a timer with a unique ID generated by `DoUniqueString`:

```
TeamFightGameMode:CreateTimer(DoUniqueString("dothislater"), {
  endTime = GameRules:GetGameTime() + 3,
  useGameTime = true,
  callback = function(teamfight, args)
    Log("Three... seconds... later...")
  end
})
```

Here, you pass in the ID of the timer (a unique string starting with `dothislater`, which can be any unique string) and an object with options. In this case, the callback will be called when the game time is greater than `endTime`, which is set to the current game time plus three seconds. Game time or server time can be used, where server time based timers will tick even while the game is paused.

CHAPTER 4

Integration with D2Moddin

Integrating your addon with D2Moddin is a relatively painless process. D2Moddin's server network supports granular versioning using simple version numbers, and normal Dota 2 addons.

See the below documentation pages to get started:

CHAPTER 5

Chat with Us

If you have any questions during development you can chat on the forums or in the irc channel **#dota2mods** on GameSurge.