GAP 301 Vadim Osipov Game Engine

# Introduction:

This is a Real Time Strategy targeted engine. Inspired by games like Warcraft 2 and Diablo 2. Where Player has one or number units to play with and number of enemy Units on the game map.

I’m sure that engine is capable of supporting side scrolling 2D game, but to be honest I’ve never tried making one in it. If you so desire.

### Tech Spec:

* **Build Platforms**: Engine can be build in x86 and x64 configurations.
* **Build Configurations**:
  + Debug - Will bring up development output console, that engine can use to log into.
  + Release - Only Game window will be shown, no development console is available.
* **Rendering**: SDL Rendering is supported.
* **Scripting**: LUA Scripting language is supported.

### Language used in Description:

* Object - Actor/Entity of the game engine, Objects can have variables, scripts with logic, and all sorts of cool stuff.
* Tile - Tile is just an image representation of a field cell.

### How do I make a game?:

It is fairly simple process, you just have to create a *main* object with script and have it do object:RequestCloseWindow() function when object is destroyed (OnDestroy())at some point or another. This will close the window from LUA script. Other the that you’re more then welcome to create as much scripts and objects as needed to go from Start of the Game to End of the Game.

# Project Set Up:

### Required Software:

* **Tiled** - Used as Level Editor for Engine.

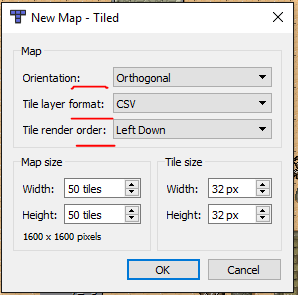
#### Good to have Software:

* **Notepad++** - Used to read and edit .json files and .lua files. Over all great free thing to have installed.

### Start New Project:

To create a new map/game in Tiled:

First create project with following configuration:



**Orientation**, **Tile layer format** and **Tile render order** are three very important settings. Please make sure to set it up the exact same way - otherwise engine will now render the map out.

Save map in **JSON** format. **JSON** is natively supported format by this engine and Tiled. You can choose the map format in Save File window from file format dropdown.

Required Media Location:

* Media Path: *\Engine\Game\Source*
* Images: *\Engine\Game\Source\Images*
* Map files: *\Engine\Game\Source\json\_files*
* Script files: *\Engine\Game\Source\scripts*

All of the folders and map file are specified in config.json file located at *\Engine\Game\Source\config* This is the only file required to be there in order for engine to work. You might change any “values” in this file. But please do not change any “keys”. *Example:* you can change map name, images folder, scripts folder.

*Note:* Please be aware of backslashes “\” they required by platform to read path.

*Note:* Please do not rename *Image File* in Tiled. Example: If you loaded *cool\_tiles.png* leave it’s name in Tiled to be *cool\_tiles*. It is one of engine dependencies. Otherwise image will not be loaded

That is it for setting up Tiled project to be this engine friendly. Next let’s set up Field and Objects

I left **tiled\_example\_map.json** file provided with engine for reference to see how I set up my development project. You may open it in Tiled and see for yourself.

### Engine Supported Controls:

All controls I tried to keep as close to classical RTS games as possible. Engine will provide you with:

* **Marquee Selection** - By pressing and holding Left Mouse Button and dragging the selection box you can quickly select and deselect groups of player Units.
* **RMB Click** - Will set “Target/Move Point” for selected Units.
* **LMB Click** anywhere on the field - Will deselect all units.
* In addition to that all input events passed into LUA Scripts as well, you’re more then welcome to add your own logic to it.

### Game Field Tiles:

**Game Field Tiles** - Images of ground tiles, just seen by the engine as just an representation of the ground images. They may carry no logic or variables, engine will simply not support it.

You can have as many layers of tiles as you like. *Note:* at the level load time all all tile layers will be flattened by the engine. So if you want to have tile on top of another tile - this will not work.

# Objects:

**Object** is a type of the object that can have logic(scripts), collision, movement component. Basically Object is something that can interact or be interacted with in the game.

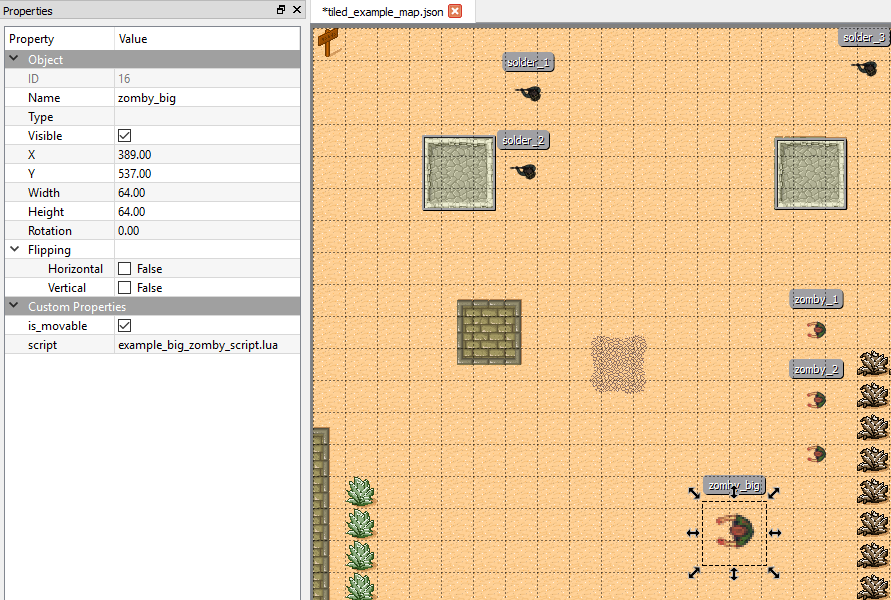
Object Variables:

* **Name**: Object needs to have a name in order to be Scripted, if it’s a simple collision box, name can be skipped.

Objects Custom Variables:

* **is\_movable** - *boolean* - lets engine know that object can be moved in the game, and should have a Movement Component on it.
* **is\_interactible** - *boolean* - lets engine know that this object should be interactible, that it can be controlled by the player.
* **script** - *string* - name of the script that object is using. More about scripting below.
* **prefab** - *boolean* - when checked: engine will not spawn an object, but instead place it to prefabs list and this object can be spawned on demand. Note: every prefab has parent, so is parent is destroyed, prefab will be destroyed too.

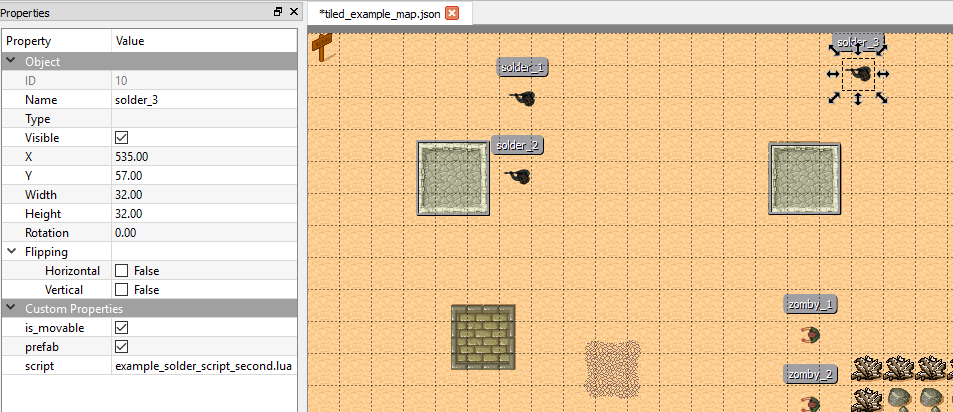
Examples:

Field Object - Spawned at load time: 

# Prefabs:

Prefab is an Object only difference is that Prefab will not be loaded on the game and engine load time, but instead Prefabs can be loaded at run time by object calling InstantiateGameObject(int posX, int posY, int prefabID) where position can be specified and what game prefab to spawn. prefabID corresponds to Tiled object ID that is specified as very first value of the object.

Example: Prefab with ID: 10 that is spawned on demand:



Note: All objects that are spawned after load time considered to be children of object that called Instantiate function. If parent is destroyed all of the children will be destroyed too.

# Scripting:

In order for object to have script it has to have *Name* and *script* custom variable with name of the script to use.

I have left a template file called template\_lua\_script.lua with all of the functions that engine will call on events, please use that template for any new script. For some reason Lua will freak out if the function is not there and you might end up with a crash or assert. Most of the functions named and commented to reflect what they do and when they are called - so I will not go into great detail about that.

As for examples: You’ve got:

* example\_big\_zomby\_script.lua - encapsulates prefab spawning logic example and movement on click. Also has example of closing window.
* example\_solder\_script.lua - encapsulates movement on Update function call.
* example\_solder\_script\_second.lua - encapsulates rotation on Update function call. Also used by example prefab in the game.

## Object Variables, a.k.a Object Data:

Every object has lua table called “values” that is updated on engine tick and has all of the most up to date values in it. Access: “object.values.*thingy*” Given values:

* **isSelected** - boolean - represents if object is selected or not.
* **posX** and **posY** - *integer* - represent object position to the window. Calculated by object actual position plus camera offset.
* **movePosX** and **movePosY** - *integer* - represent last clicked position on the screen, if object selected this can be move position.
* **rotation** - *float* - current rotation of the object. *Note*: Object original rotation is 0 unless changed directly in Tiled per object.

## Object Functions:

Functions that object can call for C++ to execute. *Access:* “object:*function(params)*” *Note:* make sure to have semi-colon sign “:” otherwise it’ll blow.

Function List:

* Log(int num) - doesn’t really work like a log function, will print out a number onto the console. Super lame.
* InstantiateGameObject(int posX, int posY, int prefabID) - Will spawn object with ID at given location. Note: Object that called this function will be a parent of newly spawned object, once destroyed - all children objects will be destroyed too. *Params*:
  + posX and posY - position where object should be spawned.
  + prefabID - ID of the prefab to spawn, yes there has to be prefab for that.
* DestroyObject(void) - Will destroy object that called it.
* RotateToVector(int posX, int posY) - Will rotate object to 2D Vector. *Params*:
  + posX and posY - positions of where to look at.
* UpdatePosition(int plusX, int plusY) - will add number to object position. Can be used to update object position on Update function call. *Params*:
  + plusX and plusY - Numbers to plus to object position, formula: currentN + plusN.
* MoveObjectToPos(int posX, int posY) - will move object to new posX and new posY. Doesn’t really work right now, I might fix it later. *Params*:
  + posX and posY - new position of the object. Object will be instantly moved.
* RequestCloseWindow(void) - Call for the Engine to close game window.

As I mentioned earlier - there are three scripts to use as reference. Don’t be shy to look inside them.

# Known Bugs:

1. Reverse Marquee selection will not work - Missing logic for reversed Rect selection. (B - Bug)
2. If Object’s child is selected and Object is not - Object will still receive OnMouseDown event. (B - Bug)