Advanced Games Programming – Report 2- Data Driven Aspects

Contents

[Introduction 3](#_Toc534625397)

[How it Works 3](#_Toc534625398)

[Why I Chose This Method 3](#_Toc534625399)

[Downsides 4](#_Toc534625400)

[Conclusion 4](#_Toc534625401)

[References 5](#_Toc534625402)

# Introduction

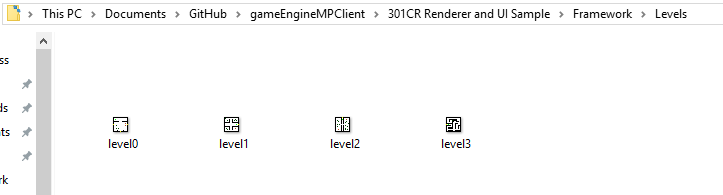
This document is designed to show the workings of the data-driven aspect of the game that I have created.

The game that I have created is a top-down ‘dungeon-crawler’, with turn-based combat and the data-driven element within it is the ability to create rooms within the dungeon using an image editor.

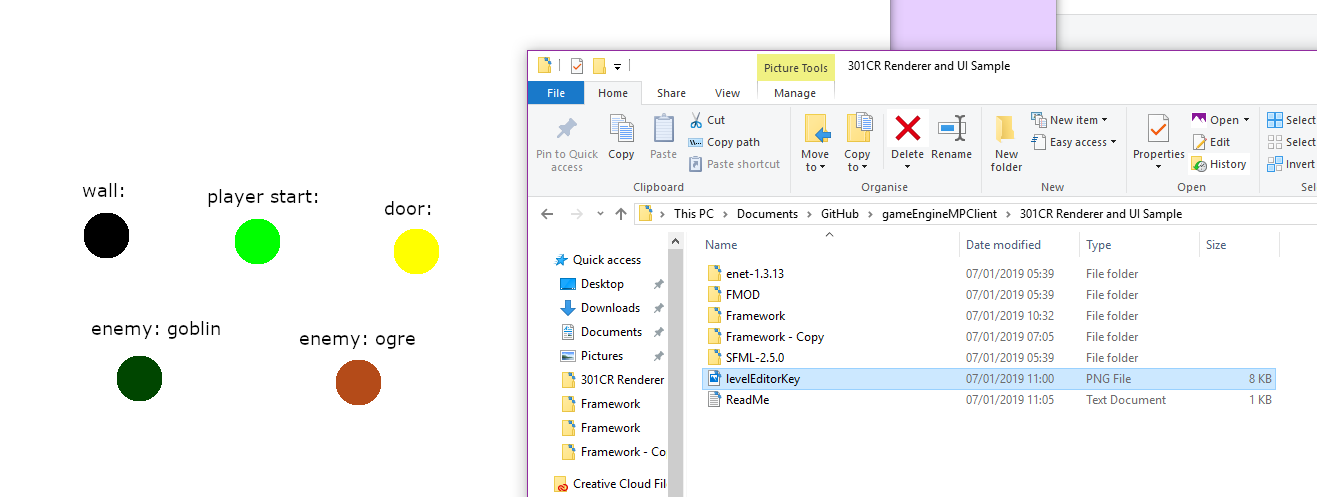
# How it Works

The level files are stored within the game directory. The game will automatically load a .png image file called ‘level0’ when it boots. So, if you wish to edit the first room of the dungeon this is where you start.

The levels are stored in the Levels folder in the directory. To edit a level simply open it in a piece of image editing software such as Photoshop or GIMP.



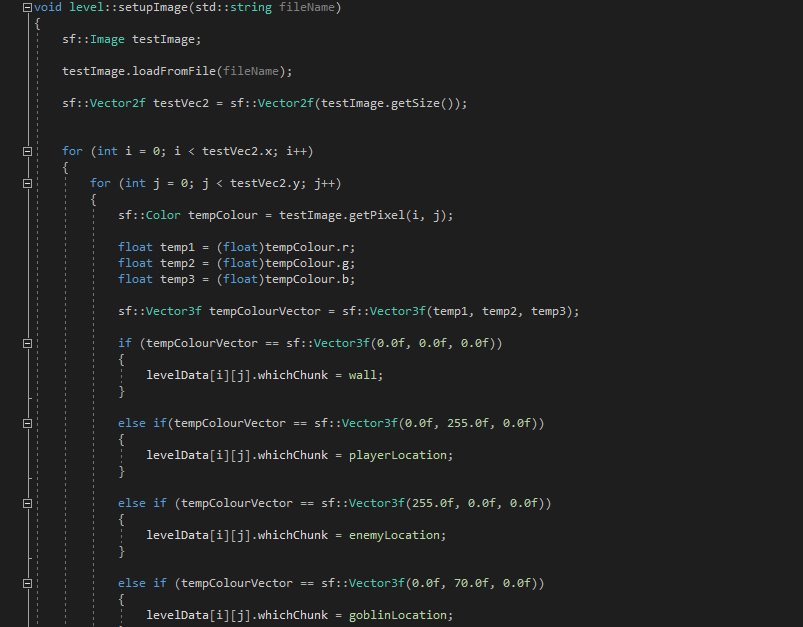
To edit the level you use pre-set colours to ‘paint’ the tiles into a level, with different colours relating to different tiles within the game. For example black is a wall, and yellow is a door. The exact colours needed to spawn each object are given within the .png image ‘levelEditorKey.png’ in the github directory. You can use then use the eyedropper tool to select the colours and then the pen set to 1 pixel wide to create new levels or edit existing one.



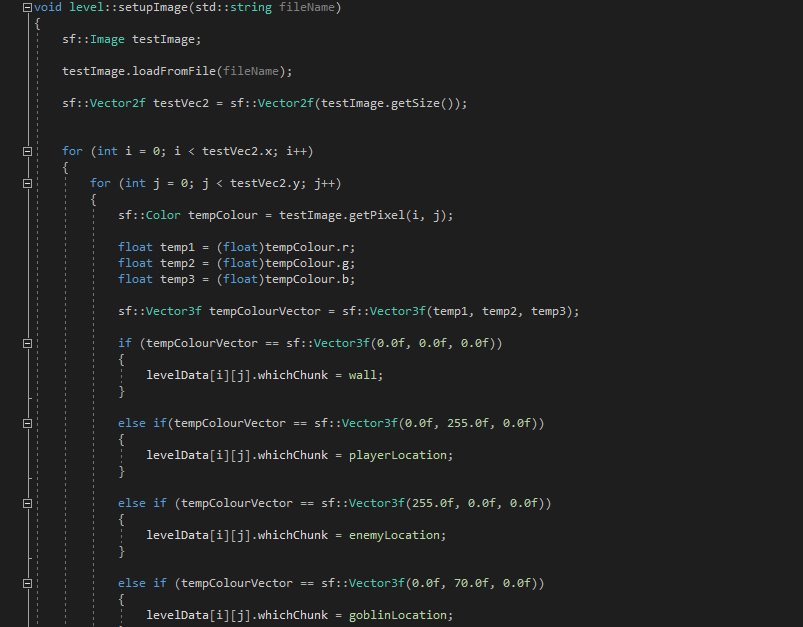
You can also place two kinds of enemy within the levels using this system. It is also easily expandable within the source code, all that’s needed is a new enemy class, a new enum within the level class to identify a new object and a new if statement within the game.cpp to add a new gameobject based on the level data.

## How it Works Within Source Code

When a new level class is created a string must be given. This is used as the filename for the level file. The level::setupImage function then loads the image based on the filename given and then looks through the image, pixel by pixel for the colours that are used, and assigns an enum in the ‘leveldata’ array for it’s position within the image.



This level data is then used within the levelSetup function within game.cpp to add game objects into the main gameobject array based on the position of the level data in the arrays and what colour the pixels in the image previously were.



# Why I Chose This Method

I chose this method as I wanted a GUI to edit the levels within my game as it is more intuitive to someone not familiar with the inner workings of the engine.

By using the pre-existing interface of a piece of image editing software I did not have to create my own GUI.

Another positive element about this approach is that many people know how to use image-editing software such as Photoshop or GIMP so many people will be immediately be able to create new levels using this data driven approach.

# Downsides

However, using this approach has negatives too. It is extremely hard to create new objects using this approach, you can currently only add pre-existing objects, so if you wanted to expand the current scope of the game you would be limited in that regard.

A potential solution to this would be to also use a scripting language such as LUA that could be used to add more custom ‘tiles’ into the game. It could do this by taking parameters such as colour and game object settings and then use these to create a new tile that could be painted on using the image editor.

You can add more tile types using the source code, however having a simpler way to perform this task would be much preferred.

Using this approach, you could expand the scope and size of the game as well as the just create levels for it.

# Conclusion

The way that I have implemented the data-driven aspect in my game, while limited in its scope is very easy to pick up and use and features a GUI, which further increases its usability.

Future improvements could greater increase the power of this system and should be considered.

# References

Sprite Art In-Game

Opengameart.org (2012) *Dungeon Crawl 32x32 Tiles*

[online] available from < <https://opengameart.org/content/dungeon-crawl-32x32-tiles>> [17 September 2012]

Sound Effects

Opengameart.org (2011) *RPG Sound Pack*

[online] available from <https://opengameart.org/content/rpg-sound-pack> [19 May 2011]