***Game Engine Development***

Target Genre

The chosen genre I have selected to make my game engine for is the Top Down shooter genre. I have chosen this genre because it allows for a wide variety of games to be created such as games in and style of “asteroids” or “bomber man”.

2D Renderer options

When deciding on a renderer for the engine I have several choices, SDL (SDL, N/A) or SFML (SFML, N/A) are the two main choices however I can also consider renderers such as OpenGL (OpenGL, N/A) or DirectX

For my game engine I have decided to use the SFML renderer, largely in part due to the high level of classes which already exists within the renderer (SFML, N/A). While SDL requires all classes to be created by a user, SFML contains useful pre-created classes for classes such as sprites and renderables. Furthermore, using SDL and building custom classes would allow for them to be more specialised and specific to the genre of game the engine is designed for, however the pre-built classes offered by SFML will help make creating both and engine and games within the engine easier as it will require less setup and use tried and tested classes. In addition, the classes in SFML can still be customised to fit the genre of game I want to make without having to be built from scratch.

Input Manager Options

This engine will need a way to managed user inputs. There a two main options for this, where the engine can either check for inputs on the object that needs them or have a dedicated polling input manager which checks all inputs and runs the appropriate functions.

While handling inputs through the objects they affect is in theory easier to implement into the engine, in practice this is not the case as it would involve multiple objects all polling for the same input separately. As such this engine will have a dedicated polling input manger which will check for inputs every frame and then link the various inputs to the appropriate processes through references to the objects which contain the functions. With a single input manager and engine only needs to check each input once per frame and then fire any functions that are needed based on the inputs, this saves CPU time and is more efficient as the manager can find and trigger any object that would be affected instead of every object checking for inputs.

I have also considered the possibility for multiple players on a single keyboard. To make the input manager work for two players I could add each player to an array upon their creation and then use the index of the array to determine which player character to call the required function on. This would allow for players to use a single keyboard and make use of the range of potential inputs.

Resource Manager

“A game’s resources must be managed, both in terms of the offline tools used to create them, and in terms of loading, unloading and manipulating them at runtime.” (Gregory, 2014) To manage large assets such as textures, file locations and pre-processing the engine will require an offline resource manager. This will handle the tasks needed to use large file sizes in the engine while minimizing the impact to performance.

In addition to this the engine contain an in runtime manager which will be the basis for “its primary mandate of loading resources into memory.” (Gregory, 2014) Such responsibilities include ensuring that only one of every unique resource, for example the scene manager, exists within the engine and managing the memory usage of the engine. This is important as without a way to manage the memory during runtime the engine will not run efficiently.

In the engine I will use a hash map in order to create the resource manager. I have chosen this method because of the high speed achievable, thus increasing the efficiency of the engine as the CPU can quickly manage its memory.

AI

For the AI of the engine I have a wide range of choices for both logic and pathfinding. For the AI’s logic system, a simple finite state machine would work best rather than a more complicated neural net or behaviour tree. For the pathfinding aspect of the AI I have a choice between A\* and a dijkstra algorithm. Both pathfinding methods yield similar results and use a grid-based system in order to find the most efficient path to a location.

For the logic systems in the AI of the engine I have chosen to use a finite state machine. While a more advanced system such as behaviour tree or neural net could be implemented into the engine I believe a finite state machine can achieve the complexity of AI needed in a 2D engine and for top down games. In a genre such as this AI objects will only need a limited number of specific states to switch between, thus making anything more complex unnecessary and complex.

Additionally, I have chosen to use A\* pathfinding for AI movement and specifically chosen to use a seeker algorithm to allow enemies to intercept and move towards and player. I believe this will be easier to implement within the engine the give a wider control over and precision of the AI’s movement. Additionally, “A\* is considered a "best first search" because it greedily chooses which vertex to explore next.” (StackOverflow, 2012)

Scene Manager

Building a dedicated scene manger into the engine would also serve it well for aspects such as time keeping and depth sorting.

Having a centralised scene manager can make the rendering of scenes more efficient as it allows for the engine easier to use by developers as it minimises the spread of rendering the scene into a single manager, rather than having to edit several locations. In addition, a scene manager also allows for a layered rendering system, wherein developers can choose what order to render their objects. This is key for 2D games as it dictates which objects appear in front of others in the renderer.

Collision Detection/ Physics

In order for the engine to detect object collision and have object physics I have the choice between the Bullet (Bullet, N/A) and Box2D (Box2D, N/A) physics libraries. Having a physics system in engine will allow for walls to limit the players movable area and create obstacles for them to move around.

For my engine I have chosen to use the Box2D physics library. I have chosen this library due to its wide list of 2D features such as continuous collision detection the multiple shapes per body. In addition, the library also has a vast documentation which will make implementing it into the engine easier. This documentation will also be useful for anybody using the engine, this will allow developers using the engine to quickly setup physics and collision systems in their games and aid in fixing any errors or problems. (Box2D, N/A)

Tool Development

In order to create the range of tools for my engine I will use a range of libraries and custom code. The main tools for the engine I will aim to make are a level editor, audio tool and a system to incorporate non-C++ languages into the engine. Adding these tools will improve the customisation and usability of the engine.

The debugger of my engine will allow creators to pause the engine at any time and inspect objects or elements of the game and gain information such as variable states and values. To achieve this, I will make use SFML’s renderable class (SFML, N/A) and present the user with a visual box displaying the information along with an output to console.

I will also add a system for users of the engine to be able to incorporate LUA code into the engines system. To create this, I will use an existing library that converts LUA code into usable information for the engine. Incorporating this into the engine will allow developers less skilled in C++ to develop games with the engines system.

Finally, I will use custom code to allow users to create their own custom maps through a notepad editor, within which each character of the pad will relate to a map tile within the engine. This will provide a simple way to build custom levels rather than the engines randomly generated levels.

# Bibliography

Box2D, N/A. *Box2D.* [Online]   
Available at: https://box2d.org/  
[Accessed 27th November 2019].

Box2D, N/A. *Box2D.* [Online]   
Available at: https://box2d.org/documentation/  
[Accessed 27th November 2019].

Bullet, N/A. *BulletPhysics.* [Online]   
Available at: https://pybullet.org/wordpress/  
[Accessed 27th November 2019].

Gregory, J., 2014. Game Engine Architecture. In: *Game Engine Architecture Second Edition.* New York: CRC Press, p. 308/309.

OpenGL, N/A. *OpenGL.* [Online]   
Available at: https://www.opengl.org/  
[Accessed 16th October 2019].

SDL, N/A. *SDL.* [Online]   
Available at: https://www.libsdl.org/  
[Accessed 12th October 2019].

SFML, N/A. *SFML.* [Online]   
Available at: https://www.sfml-dev.org/documentation/2.5.1/annotated.php  
[Accessed 12th November 2019].

SFML, N/A. *SFML.* [Online]   
Available at: https://www.sfml-dev.org/index.php  
[Accessed 12th October 2019].

StackOverflow, 2012. *Stack Overflow.* [Online]   
Available at: https://stackoverflow.com/questions/13031462/difference-and-advantages-between-dijkstra-a-star  
[Accessed 25th October 2019].