An important first step was to get Lua, SFML and Box2D building together within a single cpp project. Once the project was building correctly, the next step was to create some example classes and a simple main program loop to create a test bed for future work.

At each major stage in the development cycle testing the game engine on other platforms was ensured as a cross platform game engine is far more useful to a developer than one that is locked down to a specific platform.

Creating simple classes that showed that SFML works allowed for the creating of the first classes as shown in the class diagram. The drawable and point classes were then created as well as the surrounding framework to allow for the storing and retrieval of the objects from both Lua and in cpp tests.

The point class is purely virtual and cannot be instantiated, it provides a think function that will be overridden by all sub classes that are required to do work every cycle of the main loop. This style of programming reduces the amount of duplicated code in the core of the engine.

The drawable class provides the ability to draw a sprite at a set position and inherits from the point class so that it can be stored in the same vector as other entities and can then be casted as is required. This is a standard example of using the type theory of polymorphism [1] and allows for all subtypes to be handled by the same functions while having, potentially, fully unique behaviour.

Creating the resource manager proved to be challenging but ultimately very satisfying when it worked correctly. It allows for the end user of the game engine to not have to bother with resource management as the core engine will handle it, all resources are loaded from file for the first time but are otherwise reused by passing a smart pointer to the resource, when the resource is no longer in use anywhere it is unloaded from the resource manager.

Up to this point there was no Lua integration at all, to begin the game engines API, initialisation and a think function were provided, this allowed the cpp environment to call the init function when the lua world needed to be created and called the think function every iteration of the main loop.

User input is a requirement too so getting when mouse buttons were pressed and released and when keyboard keys were pressed and released were all provided to the Lua API in the form of hooks that were called as and when the events happened. Getting the current mouse position as well as getting the state of all buttons and keys were also provided so the end user of the engine can decide their preferred method of handling input within their

game.

Interaction with the window through SFML allows for the user of the game engine to ensure that their game works on all resolutions.

The resource manager is at the core of all file based operations in the game engine, allowing for its expansion to hold other resources has meant that the implementation of sound within the engine was trivial, using only systems that already existed.

Support for sound in Lua was then also added to allow for use of sount effects by the game developer.

The animation system must be tightly integrated into the core of the engine to ensure that there will be excellent performance passed forward.

The animation class handles all of frames, settings and timing for all animations within the game engine, having a class handling all of this allows for various systems and processes to use the same code. The UI system could use the animation class just the same as the animatable class could.

Seamless integration with Lua was required as animations are an important part of any game, ensuring that this was achieved was very difficult.

References

[1] *Polymorphism.* `http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf`.