Advanced Games Programming - 1 - Game Engine

Contents

[Introduction 3](#_Toc534637604)

[Subsystems Created 3](#_Toc534637605)

[Physics 3](#_Toc534637606)

[Audio 3](#_Toc534637607)

[Renderer 4](#_Toc534637608)

[Networking 4](#_Toc534637609)

[GUI 5](#_Toc534637610)

[Event Queue 5](#_Toc534637611)

[Asset Loader 6](#_Toc534637612)

[Conclusion 6](#_Toc534637613)

# Introduction

The game engine that I have created uses various pieces of middleware to assist with the elements of the game such as the rendering, the networking and the physics system.

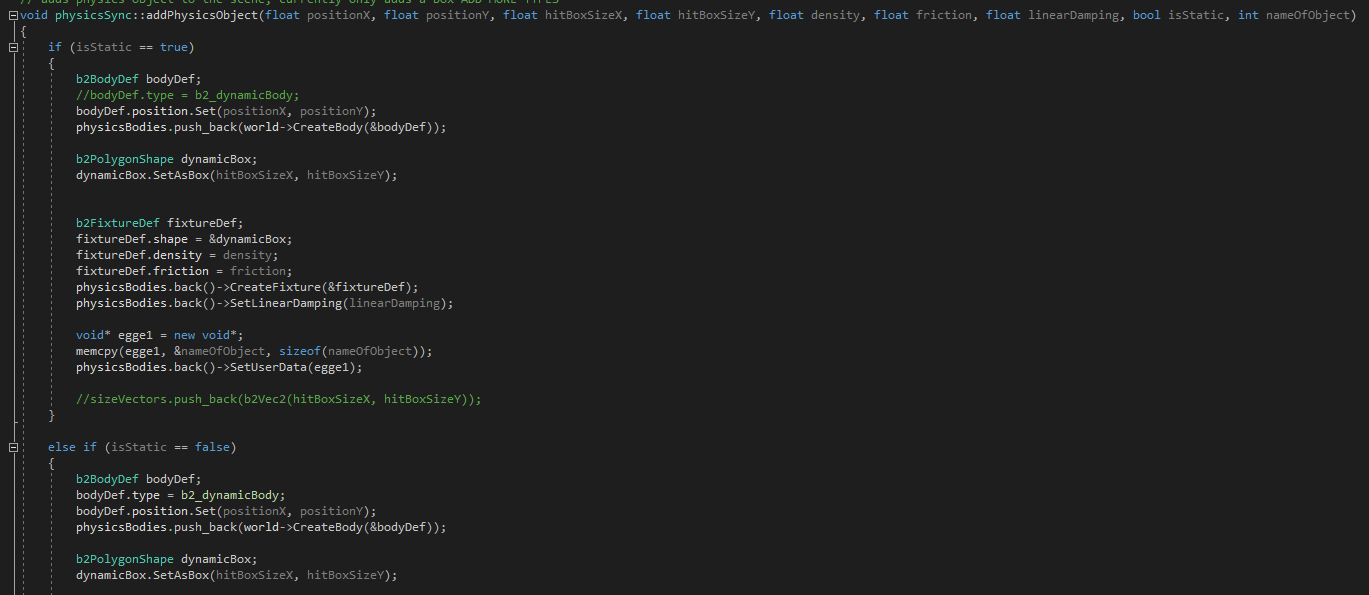
I have also used a subsystem approach to my game engine, creating individual subsystems for each major system of my game engine such as the audio, the physics and the renderer by compartmentalising these systems. This approach allows for each individual system to be replaced with an alternative if needed.

# Subsystems Created

## Physics

For the physics subsystem I used the singleton pattern. This is as there will only ever be one physics subsystem operating on one client at once. The class is only ever made once in the game.cpp.

The physics subsystem uses the Box2D physics library to handle the physics calculations for all the gameobjects that are loaded into the current scene. It does this by first populating the box2d world with box2d objects based on the parameters of the game objects that were given to it.



It also has an update function that updates the physics of all these objects at a constant rate, this allows for the physics objects to move and interact with each other.

The positions of all the physics objects are stored in a vector and then passed out, allowing for this data to be used elsewhere.

Another aspect of this subsystem is the collision listener. I have created a subclass of the contact listener that checks for any collisions to pass out data based on the game objects that have collided with each other. This allows me to have collision-based behaviours within my game.

The singleton pattern was the right choice for this subsystem as it makes sense to only have one of them active per game. You do not need more than one physics world for this application.

## Audio

For the audio subsystem I have used the audio library FMOD.

My audio system handles everything to do with audio within the game engine. It first sets up FMOD and then allows the user to load audio files into memory and then to play them with separate function.

Once again for this subsystem I used the singleton pattern. This is as I only wanted one system handling audio at once. For my engine I have basic sound loading and playing functionality playing through one channel. Having more than one audio system does not make sense for such simple functionality.

Being able to load the sounds into memory before you play them also allows you to pre-load external assets so that you do not have to load them every time you wish to use them.

## Renderer

For the rendering subsystem I have used the SFML graphical library.

The rendering subsystem takes the data of each game object in the scene and render’s these shapes to the screen accordingly. It is also responsible for the loading of font as well as the rendering of text for the gui.

## Networking

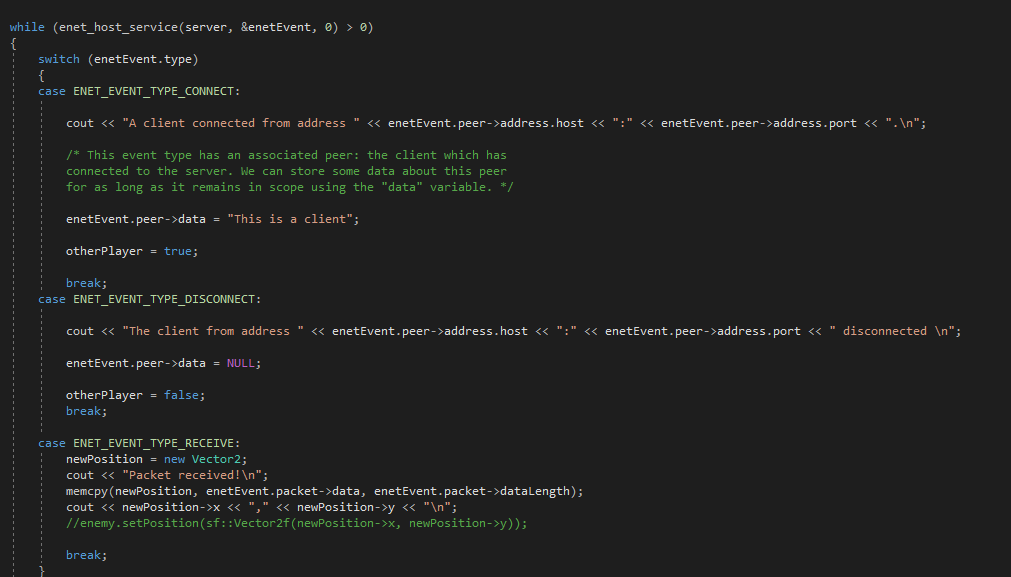
For the networking subsystem of my game engine I have used the Enet networking library.

This subsystem comes in two parts in my engine. There is the serverClient class which is used to set up a server for the game and the networkSync class that is used to connect and play with a user already running the serverClient class.

This subsystem has a variety of roles such as setting up the server itself and dealing with clients joining and leaving. It also sends and receives packets of data to and from clients to allow data to be exchanged between different players, allowing them to play together.

The networkSync class is very similar, except instead of setting up a server then handling packets in and out, it joins a server then handles packets in and out.

The networking subsystem uses the observer pattern as the subsystem is constantly checking for a change of state of the event enetEvennt such as receiving a packet of data and then reacting based on this.



This is the right pattern for this subsystem as the networking subsystem needs to be able to react to events happening within the game by other clients and needs to be able to process these when they happen.

## GUI

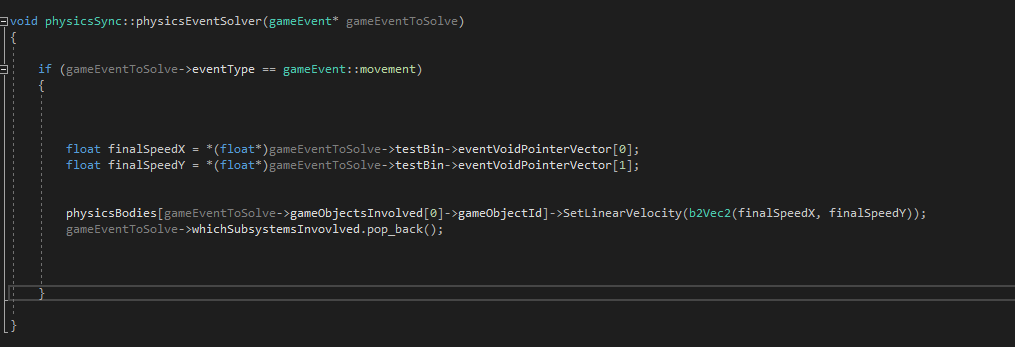
I have also created a subsystem for creating and controlling GUI’s. This subsystem allows for creation of an interactive UI. It allows you to add custom elements to it by adding a specific child of the game object class (GUIElement) as well as add text labels to objects.

## Event Queue

In my game class I have an event queue handles various game event that are generated throughout gameplay. These include moving, attacking and playing sounds.

In the physics and sound subsystems, I have handlers for these events. Once the events have been sent there from the event queue the appropriate action is performed with whatever data is given by the event.

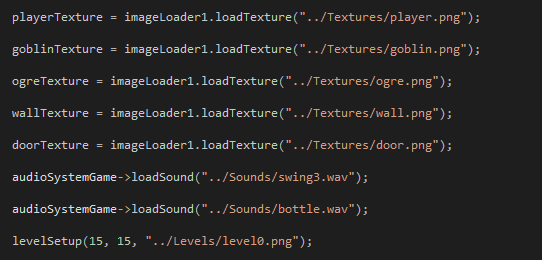
This is another example of observer pattern as the event solvers are constantly ‘watching’ the event queue for an event that they can handle and acting based upon that information.



## Asset Loader

All the external assets in my game can be loaded in before they are used, meaning that it is not necessary to load assets multiple times, saving processing time and memory.

This feature is present in both the imageLoader class as well as the audio subsystem and allows for preloading of both image and audio assets.



# Conclusion

Within my game engine I have used a modularised approach to designing the various subsystems present within, as well as how they all interact with each other. I have also considered the various design patterns when designing each subsystem.

# References

FMOD *FMOD* [online]

Available from <https://www.fmod.com/>

Enet.bespin.org *Enet* [online]

Available from <http://enet.bespin.org/Features.html>

Sfml-dev.org *SFML* [online]

Available from <https://www.sfml-dev.org/>

Box2d.org *Box2D* [online]

Available from <https://box2d.org/downloads/>