Games Engine Construction Report

Table of Contents

[**Chapter 2 User Guide** 3](#_Toc28100732)

[How to play the demo 3](#_Toc28100733)

[**Chapter 3 Known Issues** 5](#_Toc28100734)

[**Chapter 4 Implementation Check List** 7](#_Toc28100735)

[**Chapter 5 Maintenance Guide** 9](#_Toc28100736)

[Binary Tree Class 9](#_Toc28100737)

[Graphics Class 9](#_Toc28100738)

[**Chapter 6 UML Diagram** 10](#_Toc28100739)

[**Chapter 7 Conclusion** 11](#_Toc28100740)

# **User Guide**

## How to play the demo

Recognised Input - Keyboard:

* A
* D
* E
* LMB (Left Mouse Button)
* Space Bar

**A:**

This controls the movement of the player character in the left (negative) X direction and is used to navigate around the level.

**D:**

This controls the movement of the player character in the right (positive) X direction and is used to navigate around the level.

**E:**

This is an interact button and is used when the player is within range of a door to move to the level corresponding to the door (this can be the previous level or up to two different other levels.)

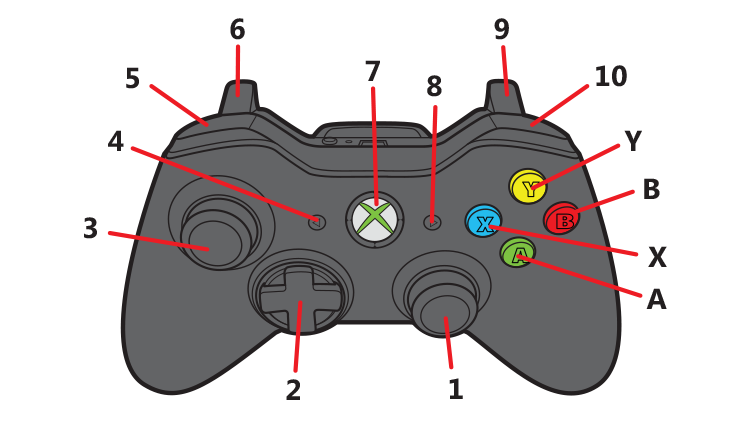
**LMB (Left Mouse Button):**

This causes the player to shoot in the direction that they are facing (this id determined based on the last lateral input used (**A** / **D**.)

**Space Bar:**

This is used to make the player character jump (negative Y) to navigate around obstacles within the level.

Recognised Input – Controller (Xbox):

* Left Analog Stick (3)
* A
* X
* Right Trigger (9)

Diagram

The diagram shows an example Xbox controller with the buttons labelled, this is to be used as a reference and the names of each button will also include the label from the diagram above.

**Left Analog Stick:**

This is an alternative to player movement if they wish to connect an Xbox controller, alike **A** and **D** is used to move the player character in the X direction.

**A:**

**A** is an alternative to the spacebar and is used to make the player character jump (negative Y) to navigate around obstacles in the level.

**X:**

**X** is an alternative to **E** that is used to interact with objects in the level such as doors that allow the player to proceed to another level.

**Right Trigger:**

This is the Controller version of **LMB** which allows the player to shoot.

# **Known Issues**

|  |  |  |  |
| --- | --- | --- | --- |
| Description of Issue | Planned Fix | Fix Difficulty | Severity |
| Moving the player character into a wall and trying to jump will not allow the player to jump (jumping when colliding results in no jump.) | This is an issue that stems from the collision system implemented, in order to fix this, I would have to be able to recognise collisions on each side of the rectangle. The fix for this would be either a grid-based collision detection and changing the way collision works to return what it collided with to determine if it is on the ground. |  |  |
| Moving a wall and shooting at the same time causes the bullets to not collide with the object and pass through it (only occurs in debug) | I assume due to the game being in debug it is not running fast enough to check the collisions and the bullets are moving through the wall before they are checked for collisions, this can be solved by using lerp on the bullet and finding if the line created intersects with any terrain or object, we then know a collision is inevitable and can set an object as collided if they are inside the collision object or past it . |  |  |
| Enemies shoot from their Model position and only in a straight line, causing bosses to be unable to shoot the player. | A fix for this would be to take the players position and the point where the bullet is spawned, and move the bullet along the vector created, this can be further advanced by predicting the next space the player will occupy by taking the players last and current position and extend the vector in the direction the player is moving by its speed and shoot there. |  |  |
| If the player changes the level while a bullet is in the air, it will continue into the next level. | Having a brief load screen to cover any screen glitches whilst resetting temporary object vectors (such as bullets) would be a good idea. This would simply be a case of setting every bullet to inactive, on a change of scene. |  |  |
| If the player moves inside an enemy and starts shooting, the collision for the bullet does not detect and passes through the enemy. | It seems that the order of collision detection may have an issue as such I will have to write a way to store all changes from the check collision function and then perform them all at once. Another fix for this would be to simply have the player collide with the enemy. |  |  |
| Always falling at max velocity | This will be a simple fix that would just reverse the jumping function in which the player slowly increases velocity until it hits a max velocity. |  |  |

# **CheckmarkCloseCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCloseCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkCheckmarkImplementation Check List**

* Randomly created level order using binary trees
* Thread pooling implemented

# **Maintenance Guide**

## Binary Tree Class

Purpose

The purpose of this class is to create a randomly generated binary tree to be used as a level structure such that each playthrough has a different order of level traversal than the last. This is done by shuffling a vector of int the size specified by the World class, then inserting those numbers into a binary tree based on the shuffled order to create a random level structure.

Re-usability

This class comes with a node class, that has no library dependencies. The Binary Tree class has three libraries that it uses: random, future and stdexcept. This allows the class to easily be imported into another project with no issues, if it is accompanied with the node class that it uses to point to a nodes parent and children. This means that simply using visual studios functionality to add an already existing file to a solution will allow the Binary Tree class to be easily added and used in any other project with a C++1x and onwards

The advantages to this are that it saves a lot of boilerplate coding and can have multiple uses as the class comes with: search, insert and delete functionality.

There are little to no downsides to this, barring from the possibility of using a version of C++ not compatible with the libraries in use.

## Graphics Class

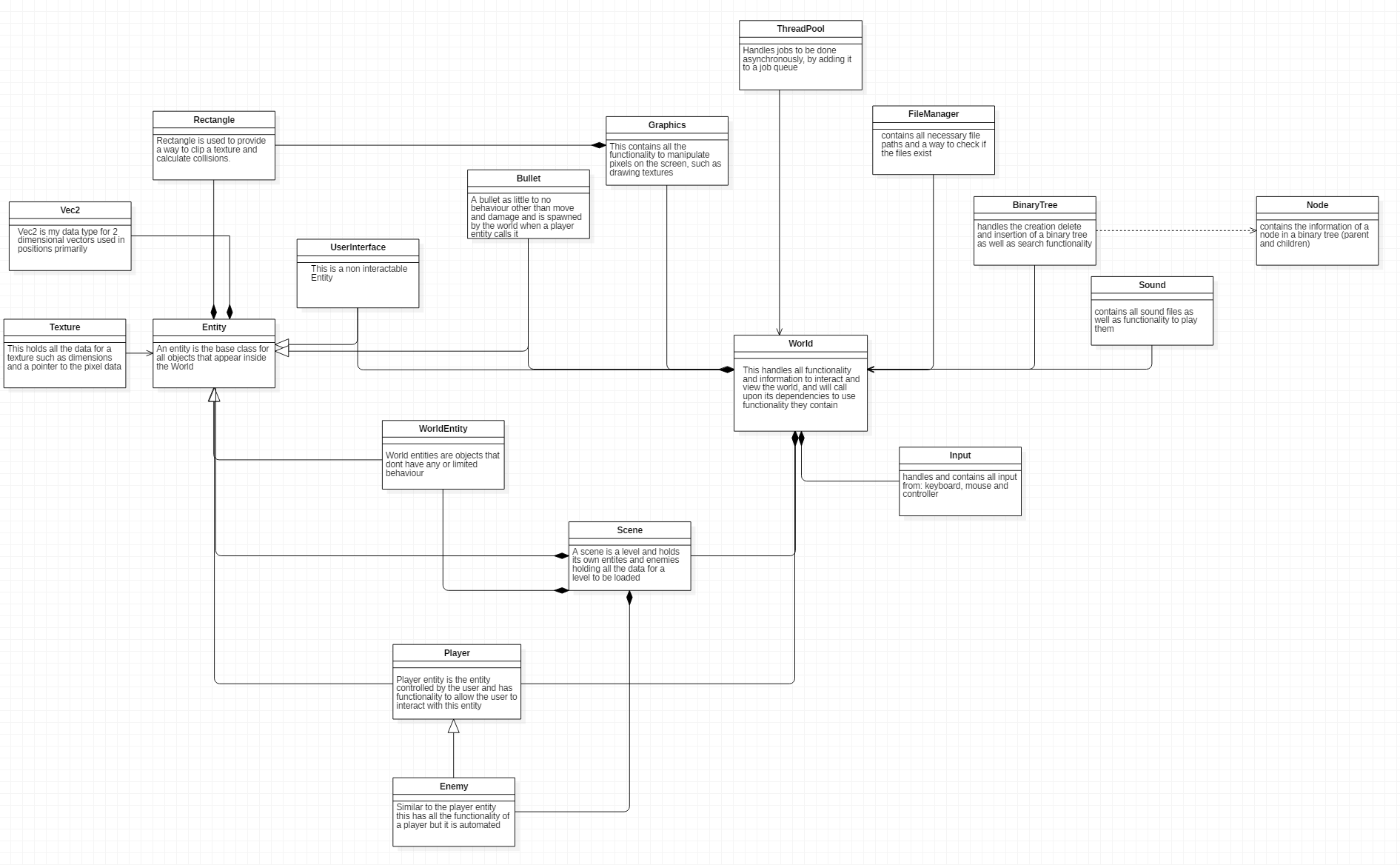
Purpose

The purpose of this class is to handle any graphics related functionality for the game this includes but is not limited to creation of the window and any interaction with the data of a pixel on the screen. For example, drawing textures to the screen that have been loaded or clearing the screen / loading a background.

Re-usability

The Graphics class is quite dependent on this solution in that it requires a Texture class and an Entity class to be in use, the texture class isn’t much more than the dimensions of the texture, a pointer to the textures and whether it is an animation or not. The entity class is quite large. The Graphics class uses the dimensions of an entity, its collision rectangle and its texture pointer, as such it would be required to bring both the Entity class, Texture class and the Rectangle class with the Graphics class for it to work as soon as it is added to another project and as such is not as re-usable as the previously mentioned Binary Tree class.

# **UML Diagram**

Below is the UML diagram for my Game Engine Demo, inside of each class is a description of its purpose and use within the demo, as well as its relationship with any other class in the solution. This has been created with a black box approach in mind; in which a World class holds all necessary information and functionality to create and interact with the world.

# **Conclusion**

Reflection

Overall, I am very satisfied with the work I have produced it has demonstrated the vast majority of functionality that was listed on the implementation checklist and has been created in such a way that it could be re-used.

There were also many mistakes within the development of this solution and the vast majority could have been solved if I had invested more time into planning the architecture of this solution, as an example my Entity base class and World class both became very bloated due to adding in some pieces of functionality halfway through development.

I have learnt now that without having a solid plan of what is needed to be done and how to do it, you can spend more time thinking of solutions to issues that are not issues in of themselves but mistakes within the organization of the solution.

Regardless, I consider this project to have been a profound learning experience for myself, although the solution itself does not show all that I have learned, it demonstrates to me how I have improved as a programmer. I have gained an expanded knowledge on modern C++ and design patterns as well as external software such as GitHub and found many more pieces of software that I will be looking into as a result of this project such as Doxygen.

With the guidance of my tutors I have learned much more about problem solving and when to consider something an issue as well as how to prioritise my workload.

In the future, as previously stated I will be investing more time into the design and planning process, as well as setting macro milestones for my work such that I will prevent myself from going a week with nothing to show for it.

More specifically, I would like to implement more templated arguments to my project, so that it is much more portable and increase my use of modern C++ and its libraries. I would have also liked to attempt to create a game with more focus on the command design pattern and implementing a debug console for better game testing and debugging. I will be considering these as milestones to further my studying from this module, as I have noticed I have improved dramatically through the course of this module due to having a set goal of creating a game engine and breaking in down in the form of the implementation checklist.