

Gavin Whitehall (S0909256)

BSc (Hons) Computer Games (Software Development)

Games Programming 3 Coursework

(MHG605291)

Module Leader: Robert Law

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.*

**Signed: Date: 16/12/2013**

Table of Contents

[1.0. Class Breakdown and Explanation 3](#_Toc374955385)

[1.1. Game1 (Main) Class 3](#_Toc374955386)

[1.2. Camera Class 4](#_Toc374955387)

[1.3. Input Manager Class 4](#_Toc374955388)

[1.4. Player Class 4](#_Toc374955389)

[1.5. Enemy Class 4](#_Toc374955390)

[1.6. Boss Class 4](#_Toc374955391)

[1.7. Projectile Class 5](#_Toc374955392)

[1.8. Functionality 5](#_Toc374955393)

[2.0. Class Diagram 6](#_Toc374955394)

[3.0. Storyboards 7](#_Toc374955395)

[4.0. References 8](#_Toc374955396)

[4.1. Development Resources 8](#_Toc374955397)

[4.2. Asset Resources 8](#_Toc374955398)

# Class Breakdown and Explanation

## Game1 (Main) Class

The Game1 class is responsible for most of the background tasks of the project; it handles a lot of functions that relate to initialisation, content loading and rendering. It also handles a number of other functions that didn’t warrant and didn’t make sense to separate into their own class. Collisions and Projectiles are an example of this as they both are required by multiple objects and classes. It didn’t make sense to create another class and create another set of instances for the Player, Projectile and Enemy objects which were already created in the Game1 class.

All the classes and objects are created for use in the Game1 class, most inside the Initialize() function (Cameras, Player, InputManager and the Enemy objects), but also some in the Fire() and EnemyFire() methods (Projectile objects) and in the SpawnBoss()function.

State management is created and handled in the Game1 class through enumeration and switch statements. The necessary values are made public so that all classes that require them have access to the current state of the game. Switch statements are implemented into every Update() function of the project as well as the Draw() method of the Game1 class to make sure only the necessary functions are updating and drawing during each state. The majority of the state changes are done through the input manager as most are for menus which are handled by input from the user.

Assets, including models, textures, sounds and fonts are all (with the exception of one, which will be discussed in the Enemy Class sub-section) loaded in the LoadContent() function. The LoadSound() function was created to separate the loading of sound assets to keep the code neat and readable, the function is called from LoadContent(). SetupEffectTransformDefaults() is called for every 3D model in the scene upon the asset being loaded, this method takes the asset and applies the view, projection and world matrices to it.

The Update() method is used to call the update functions for all the projectiles in each of the Player and Enemy lists and to call the Collisions() function. It also updates the position of the sound listener once per frame to the player’s position.

The Draw() method handles all the rendering for every object in the scene. All the 3D models in the scene have an updated transform (based on their current scale, rotations and position) applied to them and are rendered once per frame. Any GUI (Graphical User Interface) objects (2D textures and text) are also rendered here and have been coded in relation to the size of the screen, so that they automatically scale. The DrawModel(), WriteText(), DrawGUI(), DrawLaser() and DrawSkyBox() functions are all called from the Draw() method and are keep the class simple and readable.

## Camera Class

The Camera class is responsible for creating a View and Projection Matrix used for rendering the scene upon initialisation. It has a number of public members that link to the private variables that are useful for other classes to utilise, including the Vector3 Position, Vector3 LookAt and Vector3 Up variables.

## Input Manager Class

Input Manager is split into three separate functions: KeyboardInput(), MouseInput() and ControllerInput() which once per frame check for changes in input from the Keyboard, Mouse and Xbox 360 controller respectively.

Upon input registration the Input Manager has the access to change variables and call functions in other classes (Game1 and Player) to provide functionality like changing the player speed (Boost), rotating the player and muting the audio.

## Player Class

The majority of the logic and functions behind the player controlled ship are located in the Player class. The class handles all the functions and maths behind the movement and rotations of the player ship including most of the logic for the camera objects movement and rotations. The player’s health and boost functionality is also coded into the Player class, and the condition upon either reaching zero is also handled within this class.

## Enemy Class

The Enemy class encompasses the Artificial Intelligence (AI) which governs what an enemy ship does within the game. The class follows the same function structure for the movement as the Player class and these functions are called when necessary by the AI. Upon the player coming within a set distance of the enemy the Attack() function is called where the enemy will begin to chase the player through Follow() and fire upon them.

As previously stated the Enemy class is the only instance in which an asset is loaded, this was a fix for an issue that would on occasion happen. As this class is created in the Initialize() function of the Game1 class it is completely iterated at this point. This on occasion lead to the error that, on the condition that an enemy spawned close to the player it would attempt to play the Sound before the LoadContent() in Game1 had even started. The decision was made to load the sound upon creation as a quick fix to this.

## Boss Class

The Boss class is structurally very similar to the Enemy class minus movement. The boss object will be created upon the list of enemies reaching zero, when all enemies are destroyed, and will also create a user interface element. This is to show the player how much health the boss ship has left as it requires the player to hit 50 lasers in order to complete the game. On the condition the boss has been destroyed the game state will change to the completed screen where the player can then exit.

## Projectile Class

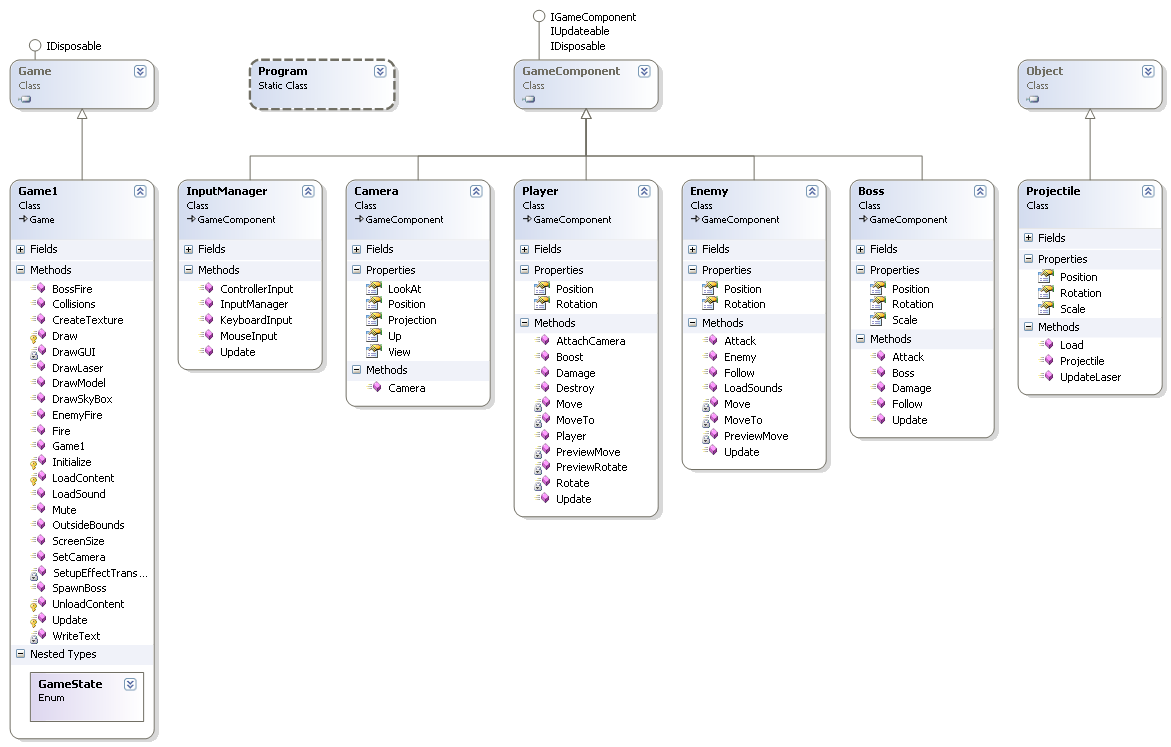
All lasers (projectiles) are created from the Projectile class, which upon creation is given a spawn position and rotation relative to the ship creating them and is translated along its local z-axis. The projectile is fed a Boolean value from its creator to specify which gun turret of the ship it is to fire from, so it may adjust the offset value which sets it the position in front of the correct turret. A Boolean value is also passed in relating to whether the ship firing this projectile object is an enemy ship or the player ship, so the projectile class will play the appropriate sound effect. The final Boolean passed to the projectile relates to the boss class and whether the projectile is called from the boss ship. This will give the projectile an increased scale and speed. The UpdateLaser() handles the frame by frame functions for the Projectile class and is called from the Update() function in the Game1 class for all the active lasers in the scene.

## Functionality

The following is a list of features and functionality that are out with the specification for this coursework project:

* Input Manager Class
* Xbox 360 Controller Input
* Camera Class
* Artificial Intelligence
* Terrain and Skybox
* State Management and Menu System
* Energy and Health User Interface
* 3D Sound Effects

# Class Diagram



# Storyboards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C:\Users\gwhite29\Desktop\1.png |  | C:\Users\gwhite29\Desktop\2.png |  | C:\Users\gwhite29\Desktop\3.png |
| Menu - Start, Controls and Exit Selection |  | Input - Controller, Mouse and Keyboard |  | Player flying around Terrain |
| C:\Users\gwhite29\Desktop\4.png |  | C:\Users\gwhite29\Desktop\5.png |  | C:\Users\gwhite29\Desktop\6.png |
| Enemy Chasing the Player, Inverted Camera |  | Boss after all Enemies are Destroyed |  | Game Over Screen upon Completion |

# References

## Development Resources

http://www.xnadevelopment.com/

http://www.riemers.net/

http://www.rbwhitaker.wikidot.com/

http://www.msdn.microsoft.com/

http://www.stackoverflow.com/

http://www.gamedev.net/

http://xboxforums.create.msdn.com/forums/

http://www.youtube.com/user/oyyou91/

http://www.youtube.com/user/Direct3DTutorials/

http://www.gamedev.stackexchange.com/

http://www.xnawiki.com/

http://www.xnagameprogramming.blogspot.com/

http://www.toymaker.info/

## Asset Resources

http://www.scifi3d.com/

http://www.turbosquid.com/‎

http://tf3dm.com/

http://www.galaxyfaraway.com/

http://www.sa-matra.net/sounds/starwars/

http://soundfxcenter.com/

http://starwars.wikia.com/wiki/Main\_Page/

http://en.wikipedia.org/wiki/Star\_Wars/

http://www.thenerdcabinet.com/

http://www.youtube.com