CGP600 – ASSIGNMENT 1

ANALYSIS AND DESIGN BY **JAKUB SIEWNIAK** AND **KANAGA MANIKANDAN SOLAIKANNAN**

REPORT BY **KANAGA MANIKANDAN SOLAIKANNAN** – Q12389501

20 OCTOBER 2017

SOUTHAMPTON SOLENT UNIVERSITY

**CONTENTS**

[Introduction: 2](#_Toc496101521)

[Game Mechanics: 2](#_Toc496101522)

[Player Movement: 2](#_Toc496101523)

[Combat: 2](#_Toc496101524)

[Entity AI: 2](#_Toc496101525)

[Survivability: 2](#_Toc496101526)

[High Score Ranking: 2](#_Toc496101527)

[Lose Condition: 2](#_Toc496101528)

[Group Design: 3](#_Toc496101529)

[User Stories: 3](#_Toc496101530)

[Allocation of Tasks: 3](#_Toc496101531)

[Class Diagram: 3](#_Toc496101532)

[Critical Path Analysis: 3](#_Toc496101533)

[Testing Plans – Kanaga Manikandan Solaikannan 3](#_Toc496101534)

[My Tasks: 3](#_Toc496101535)

[Assets: 3](#_Toc496101536)

[Static Objects: 3](#_Toc496101537)

[Dynamic Objects: 4](#_Toc496101538)

[Collectibles: 6](#_Toc496101539)

[Particles: 7](#_Toc496101540)

[Camera: 7](#_Toc496101541)

[User Interface (UI): 8](#_Toc496101542)

[Lighting: 10](#_Toc496101543)

[Development Techniques: 10](#_Toc496101544)

[Object Oriented Design: 11](#_Toc496101545)

[Critical Reflection: 12](#_Toc496101546)

[References: 12](#_Toc496101547)

**LAST BREATH**

# Introduction:

Last Breath is a third-person survival game that takes place on a highly secured island and the player is trapped in it. The player will have to find weapons, food and kill guards to survive on the island for the longest time possible. This document is a software design of the game.

# Game Mechanics:

## Player Movement:

The player will be able to walk or run around the environment and jump on small platforms to reach higher areas.

## Combat:

The player will be able to use melee weapons found in the island to fight against guards patrolling the area.

## Entity AI:

The enemies will patrol specific areas and engage on combat once the player is in their sights. The animals simply wander around the island and flee when attacked by the player.

## Survivability:

The player will have to hunt for food or kill enemies and loot food from the corpses to stay alive.

## High Score Ranking:

The player’s rank in the leaderboards is based on both, the score earned by killing enemies and the time survived.

## Lose Condition:

The player will lose the game once it loses all its health.

# Group Design:

## User Stories:

## Allocation of Tasks:

## Critical Path Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
| NODE | ACTIVITY | DURATION (HOURS) | MEMBER |
| A | Design World | 6 | Mani |
| B | Design Static  Objects | 1 | Mani |
| Design Dynamic Objects | 1 |
| C | Design Input System | 3 | Jakub |
| D | Design Player Movement | 3 | Jakub |
| Design Player Health and Energy | 1 |
| E | Design Entity Movement | 3 | Jakub |
| Design Entity Attributes | 1 |
| F | Design Player to Object Collision | 2.5 | Jakub |
| Design Player to Entity Collision | 2.5 |
| G | Design Enemy AI | 2.5 | Jakub |
| Design Animal AI | 2.5 |
| H | Design Camera Behaviour | 3 | Mani |
| I | Design Lighting | 5 | Mani |
| J | Design Physics system | 5 | Mani |
| K | Design Particle System | 4 | Mani |
| L | Design HUD | 2 | Mani |
| Design Main Menu | 2 |
|  |  |  |  |

## Testing Plans – Kanaga Manikandan Solaikannan

Testing for this game is going to take place after the implementation of every game element. Using Table 1., I will able to list down all the features and check if they’re working as per expected.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Task | Expected Result | Actual Result | Status (Pass/Fail) | Notes |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

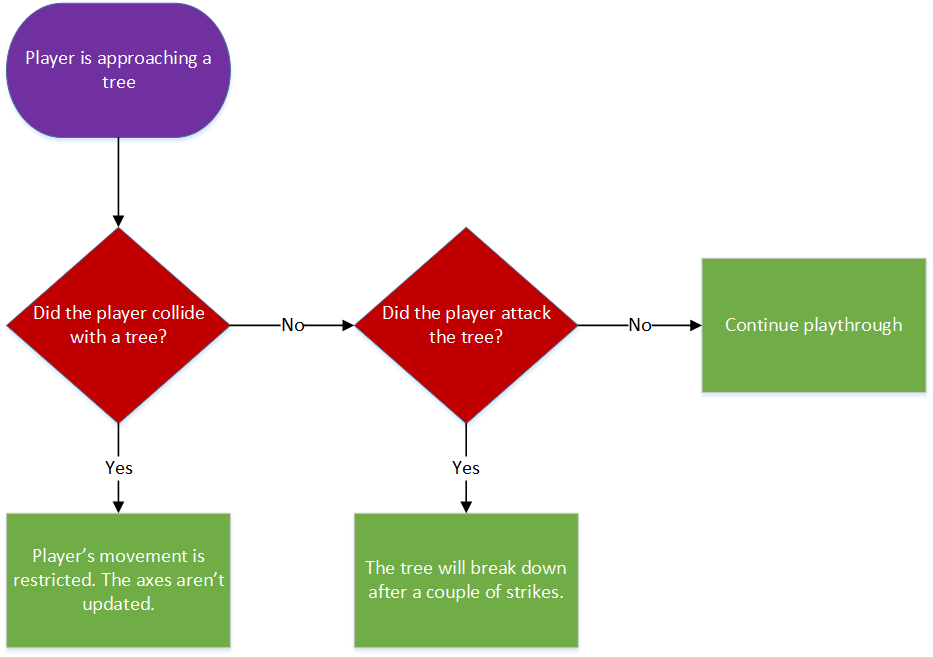
**Table 1. Testing Table**

# My Tasks:

# Assets:

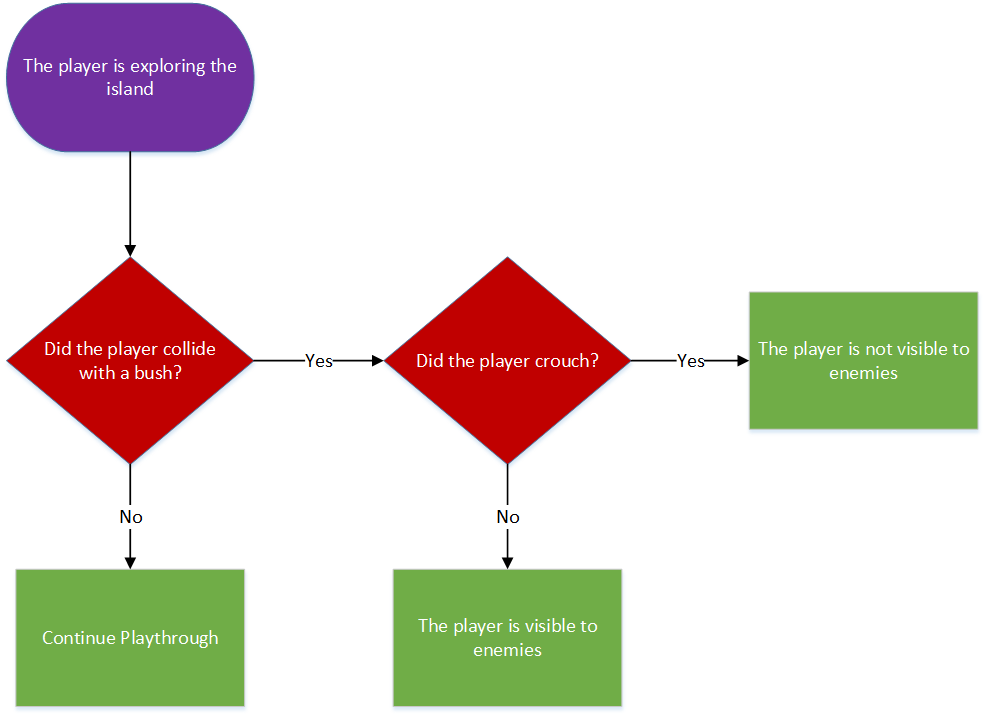
## Static Objects:

1. **Trees:** The stem of the tree will be in brown cylindrical shape and the leaves will be in a spherical shape placed on top of the stem. The player’s movement is restricted when it collides with a tree. The player can also cut down a tree to obtain a weapon. A clear depiction of this mechanic is shown in Figure 1.



**Figure 1. Flowchart of Tree Mechanic**

1. **Bushes:** These will be scattered around the island as green cubes. The player will be able to walk through it and crouch to hide from enemies’ line of sight. A visual representation can be seen in Figure 2.

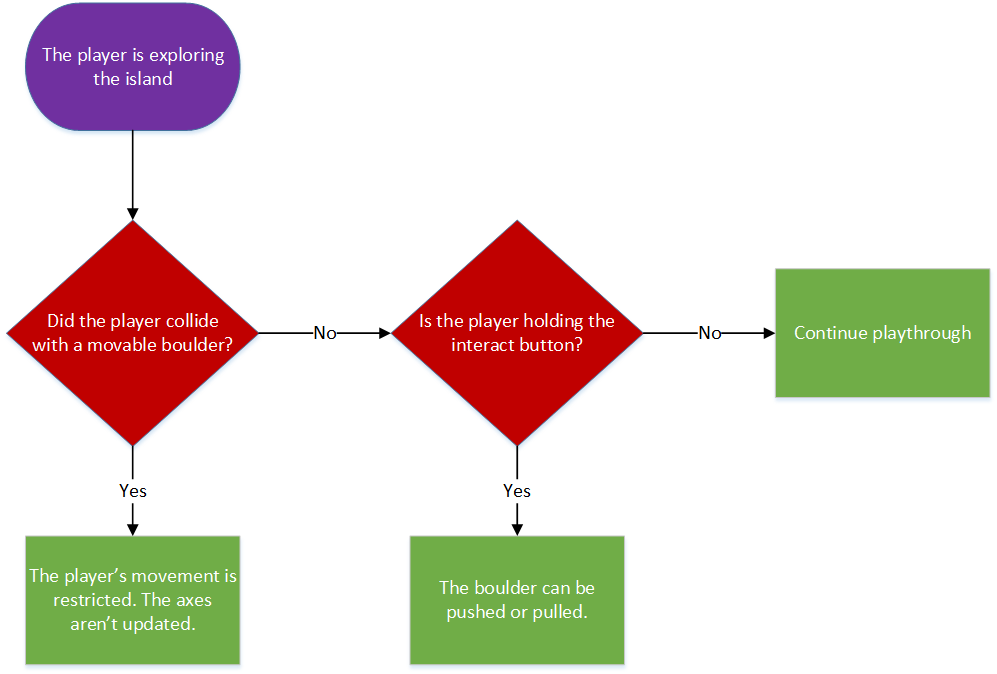


**Figure 2. Flowchart of Bush Mechanic**

1. **Boulders:** These are also scattered around the island as black cubes. They simply exist like walls and will restrict player/entity’s movement axes when collided with.

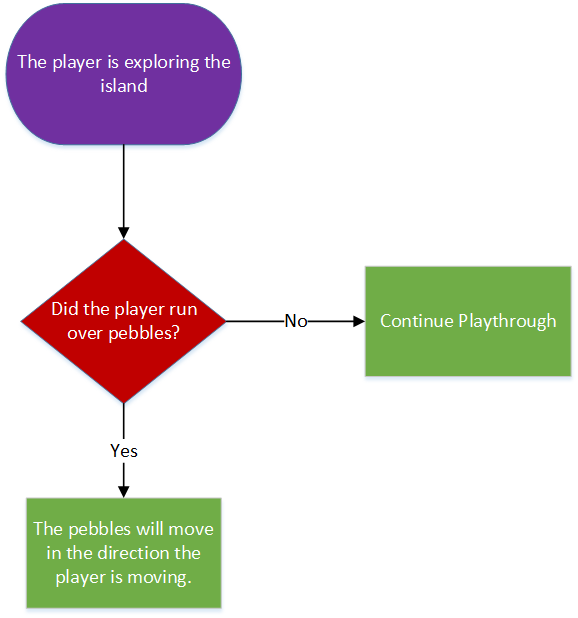
## Dynamic Objects:

1. **Movable Boulders:** These can be distinguished from stationary boulders by their lighter black colour. They are also in the shape of cubes and can be moved by the player. The boulder’s position will mimic the player’s position when moved. The mechanic is visualised in Figure 3.



**Figure 3. Flowchart of Moving Boulder Mechanic**

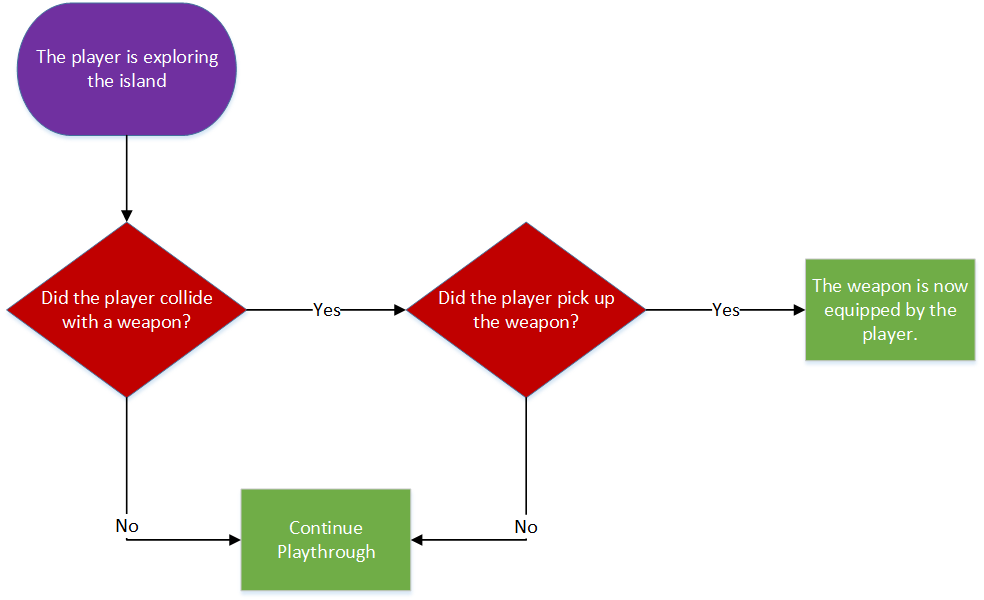
1. **Pebbles:** These are represented by tiny black spheres that roll over the ground when the player walks on it. A small force is applied to these pebbles upon collision that allows them to roll in the direction the player is moving.



**Figure 4. Flowchart of Pebbles Mechanic**

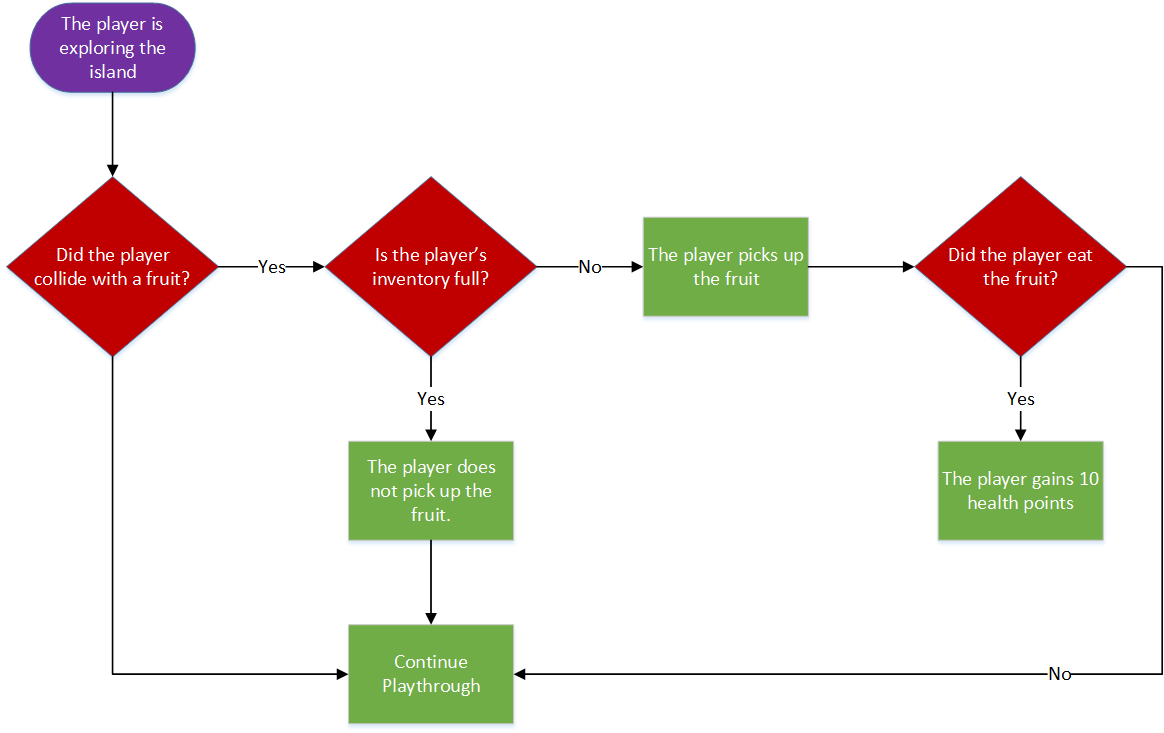
## Collectibles:

1. **Weapons:** The weapons are identified as long brown colour cuboids which are also scattered around the island. These can be used to attack soldiers or hunt animals. Once the player has picked up the weapon, the weapon’s position will be set to the player’s hand position and will update the HUD with the durability of the equipped weapon. A visual representation of the mechanic can be seen in Figure 5.



**Figure 5. Flowchart of Weapon Pick-up Mechanic**

1. **Food:** Foods are found as apples and they’re going to be red and spherical in shape. The player will be able to carry a maximum of 3 apples and they regenerate health when eaten. Health and the number of applies are going to be calculated as integers. The player will automatically pick up the fruit if its inventory is not full. The mechanic is shown clearly in Figure 6.



**Figure 6. Flowchart of Food Mechanic**

## Particles:

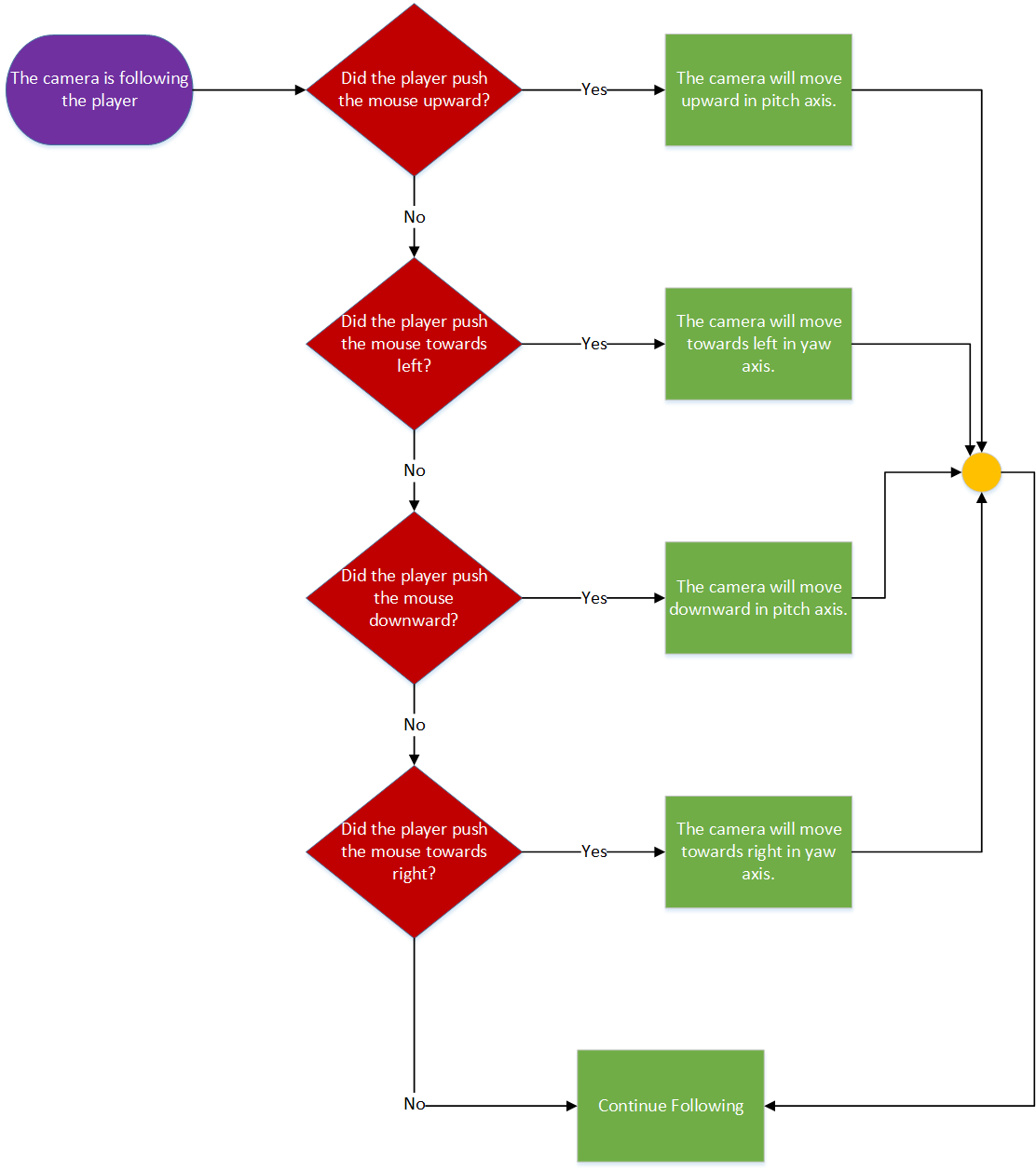
1. **Leaves:** Leaves are sometimes found to be falling from trees or blown by the wind. They’re going to be represented as flat thin cubes. A particle class will be created to load a single leaf asset and add some texture to it. Another class will be created to spawn multiple instances of the leaf created by the previous class and have different methods for random movement.

# Camera:

Since this game is played in third person, the whole body of the player will be seen and will follow the player within 5 meters or so.

The camera’s position is updated along with the player’s position and this will ensure that the camera is following the player always. This is performed by calling the XMMatrixLookAtLH function from the XNA Maths library which takes the camera’s position, target position and the direction that faces up as parameters and setting this matrix to the view matrix in the Render function (1. Sherrod and Jones 2012, p.328).

The camera can also be rotated around the player in pitch and yaw directions. This action is executed by calling the XMMatrixRotationRollPitchYaw function from the same library that takes the rotation values for all 3 axes as parameters (2. Sherrod and Jones 2012, p.338). The user can move the camera using the mouse as shown in Figure 7.



**Figure 7. Flowchart of Camera Mechanic**

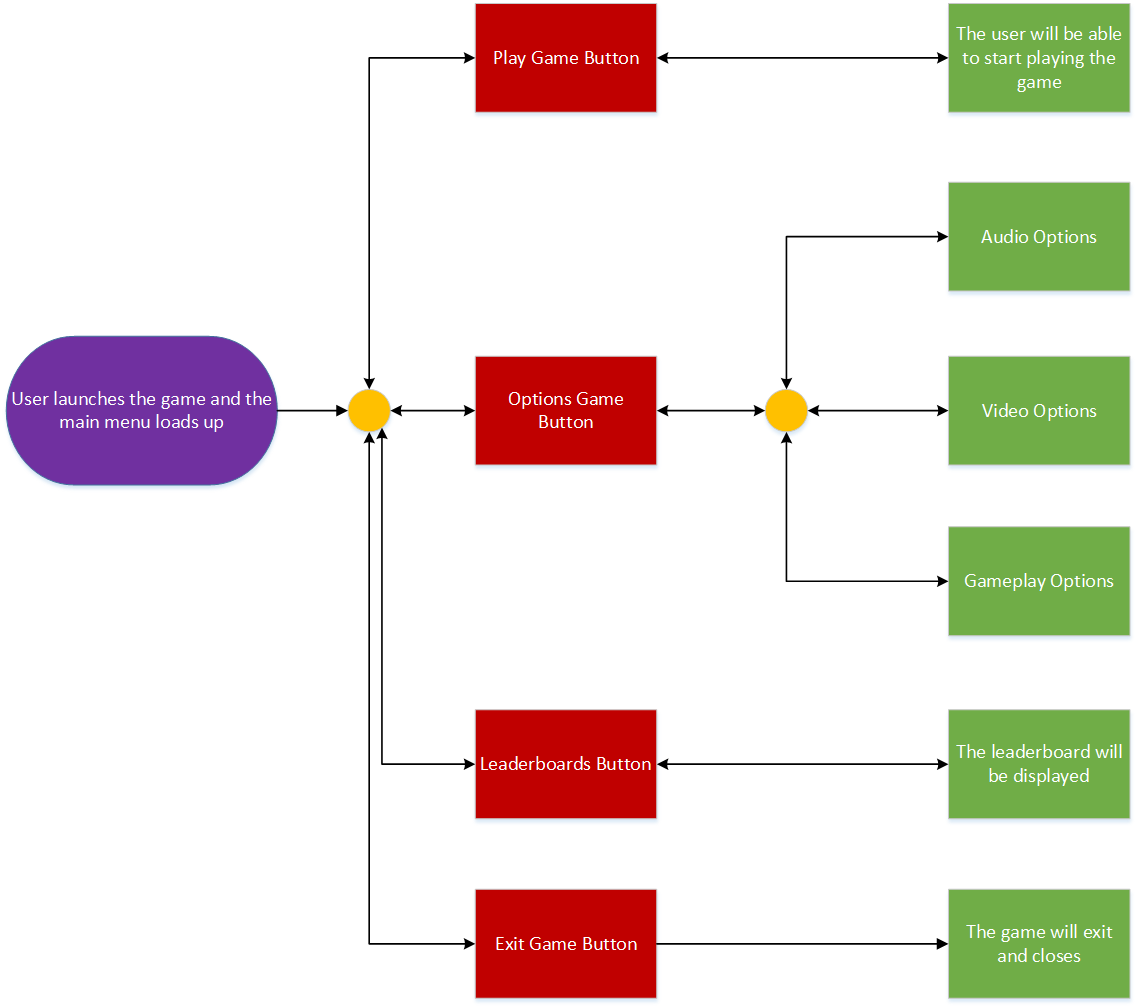
# User Interface (UI):

The main menu is going to consist of a play button, an options button, a leaderboards button, and an exit button. The play button will allow the user to enter the game and start playing. The options button will allow the user to adjust technical features such as sound, shadows, field of view (FOV), etc. The leaderboards button will allow the user to view the scores earned by players and for how long they have survived. The exit button will simply exit the game.

A click on the button will call a function the button is referring to and the function will execute the appropriate action(s). The options menu is going to have different pages for different settings and all these pages will be categorised as functions which are responsible for modifying the settings. During the process of transitioning from one page to another, a separate function will be called to stop displaying the current page and display the new page. A visual representation of the transition between different pages of the main menu can be seen in Figure 8.

The in-game heads up display (HUD) will contain the player’s health, its energy, the equipped weapon’s durability, the number of fruits carried, the score earned and the time elapsed. Each UI element will receive their respective info from various classes and display them spontaneously.

All the text in the game is going to be rendered by using each letter as assets (3. Sherrod and Jones 2012, p.162).



**Figure 8. Flowchart of Transition in Main Menu**

# Lighting:

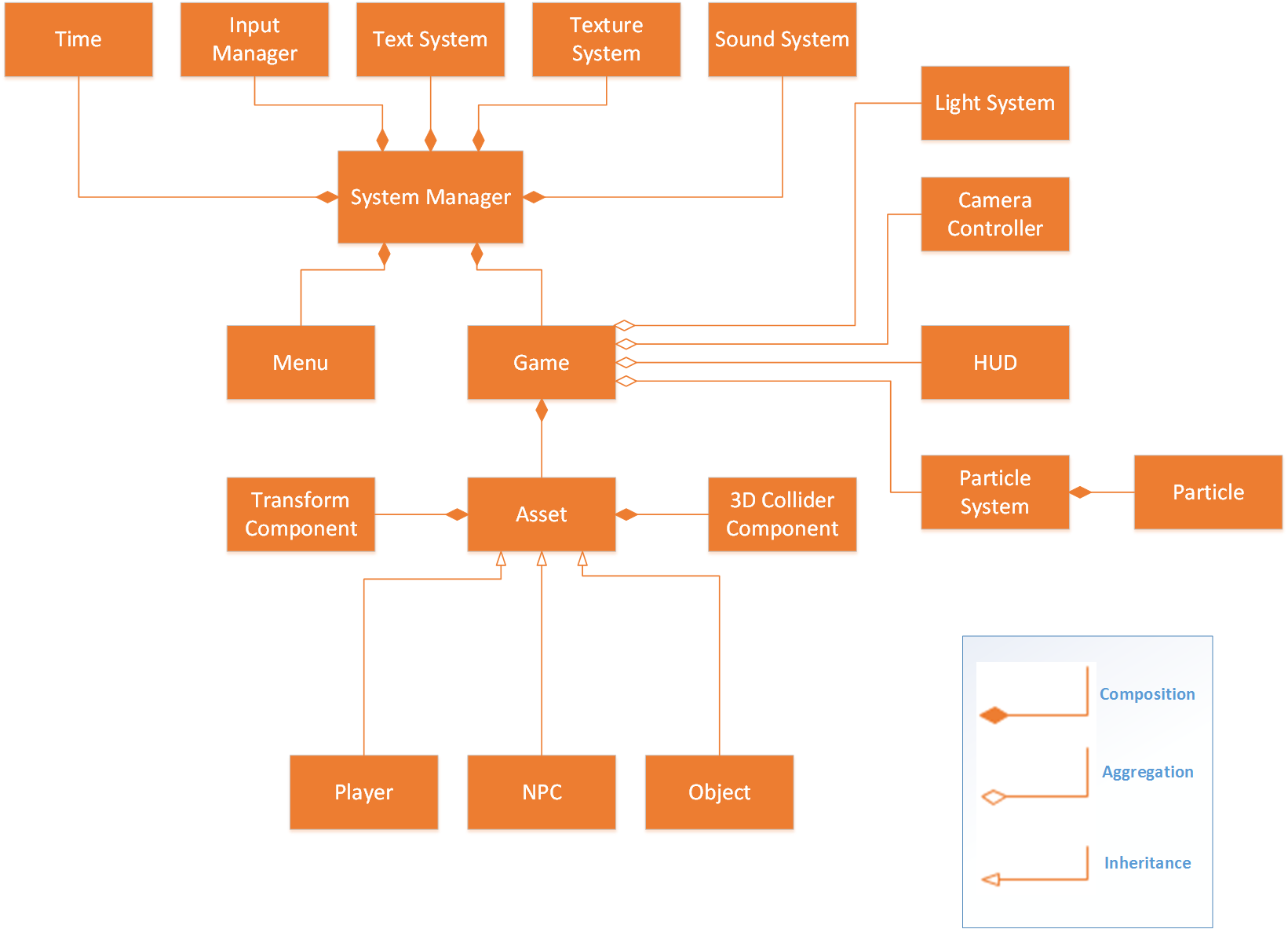
The game is going to consist of a single light source which is the sunlight. It is going to illuminate almost the whole part of the island with a bright yellowish-orange colour. The process of simulating light is a slightly complex process but to describe it in simple terms, light consists of 3 terms: ambient, diffuse, and specular. We’ll being using all these terms to produce light in our game and I will now explain how each term is going to be calculated.

The ambient term consists of just a single colour which can be added to the final colour. The diffuse term is determined by calculating the dot product of the light vector and the normal of the surface. The specular term is calculated by the value of the dot product raised to the power of a certain integer. The final colour of the light is established by multiplying all the 3 terms (4. Sherrod and Jones 2012, p. 318-324).

# Development Techniques:

To fully understand each concept and logic of DirectX 11 SDK, the tutorials for each aspect will be completed first. Based on the knowledge gained from each tutorial, an empty project will be created which is going to be the final game. The empty project will be populated frequently after understanding each concept. The world will be created first which will then be filled with objects and then with living entities. The collision detection and AI will be the next thing taking place followed by lighting, UI, etc. The final phase would include optimisations, memory management, smoother frame rates, bug fixing, etc.

# Object Oriented Design:



**Figure 9. Class Diagram**

Referring to the class diagram in Figure 9., System Manager is the primary class. It contains and initialises all other classes when the game is launched. Each class’s purpose will now be explained briefly. The Time class is responsible for receiving the system time and calculating the time difference between each tick. The Input Manager is responsible for receiving the input from the keyboard and mouse and passing the information to other classes that need it. The Text System class will load in all the text assets and can be used for UI and such. The Texture System class is responsible for loading the textures from the system folder into the game. The Sound System class is responsible for initialisation and import of sound files from the system folder into the game.

Menu and Game have separate loops that will run depending on the user input. The menu loop will run first by default displaying the main menu and when the player wishes to enter the game, the game loop will commence from the Menu class and the reverse happens when the player wishes to go back. Moving on to the children of the Game class, the Light class is responsible for projecting light in the game and cast shadows appropriately. The Camera Controller class is responsible for the movement and rotation of the camera based on the information received from the Input Manager class. The HUD class is responsible for displaying the text and player info during gameplay. The Particle class doesn’t have any actions but contains info and dimensions of a single particle. However, the Particle Manager class is responsible for generating many instances of the single particle and manage its randomised movement. These children have aggregation relationship towards the Game class because they can exist independently but they are not allowed to have other parent objects.

The Asset class is responsible for loading in assets from the system folder into the game and carries position and scale info of that asset. The Transform Component class includes all the transformation matrices required for the assets to move around. It takes in from the Asset class and executes translation and rotation functions with those values. The 3D collider component class will provide collision boxes to assets and check for collisions every frame. The player, NPC and Object classes are just used to control their actions and behaviour. These 3 classes are inherited from Assets class because these classes utilise the values from the parent class. The remaining classes have composition relationship because the children classes can’t exist without the parent class.

# Critical Reflection:

It was an amazing experience in designing this project. DirectX 11 is an interesting subject and having done some tutorials and research, I’m sure that it’s going to be fun coding with this SDK. The concepts and the logic was much easier to grasp compared to other SDKs. It’s a recommended software to any beginners to programming.

DirectX 11 excels in 3D programming and we were asked to make a 3D game using the DirectX 11. This was also my first time coding a playable game in 3D or else it was just the game engines that did most of the 3D work. So, I should say I was very much excited about this project. My partner and I sat down and thought of what kind of game could we make. We didn’t want to make a very complex game mainly because it wasn’t required. That’s when we came up with this island survival game because it was required to make a large area for the player to explore.

We designed our game by constantly referring the assessment brief and making sure if our project has met all the requirements. We weren’t sure which tasks were complicated and which ones were simple. So, we faced a bit of dilemma on how to fairly distribute the tasks between us. We, first, split the tasks based on our programming experience such that we knew that implementing UI is easier than implementing collision detection. This wasn’t a permanent solution but it was a good lead on at least starting our individual reports. We, later, came up with a points system that can be provided to each task to identify its difficulty level. This made it easier for us to distribute the tasks between us evenly as shown in [ENTER HERE]. My partner and I couldn’t find time outside university hours to discuss and design our project. So, we spared time when both of us were at home and started video calling through Skype. He could share his screen and we were able to get the job done conveniently.

Now about my individual tasks, I should say I was able to design easily with a help of a DirectX 11 book for beginners. The book stated everything I needed to know about the tasks allocated to me and was very informative.

In all honesty, there wasn’t much difficulties faced in designing this project. One of the reasons could be that I designed game projects before and it was a similar process.

# References:

1. SHERROD, A. and W. JONES, 2012, p.328. Beginning DirectX 11 Game Programming. USA: Course Technology.
2. Ibid., p.338.
3. Ibid., p.162.
4. Ibid., p-318-324.