Distinction Project Plan

# The Plan:

***Game:*** Lunch Break

***Engine:*** Unity (<https://unity3d.com/>)

***Language:*** C#

I plan on using my capstone final year game project as my Distinction project game.

The game, *Lunch Break*, has been in development since semester 1 and I am part of an 8 person team (5 designers, 3 programmers) who are bringing this game to life. The game is being developed in Unity with all of our scripts written in C#.

I plan on using the code and C# scripts I have written for the game to show off techniques that I have used which relate to the techniques and ideas learnt in this subject. These bits of code will relate directly to the intended learning outcomes in this unit and demonstrate my understanding of many of the ideas from games programming.

As an addition to writing about what parts of code were done well, I will also be discussing what parts of the code didn’t work so well or could have been better achieved had I done this subject before making the game.

# The Game and the ILOs:

In this section of the report I am going to go over a few features of the game, the code used for those features and how they relate to the Intended Learning Outcomes for this unit.

The design and implementation ILOs (1 & 2) refer to the use, design and implementation of game engine components. Since *Lunch Break* was developed in the Unity game engine which uses a component based system for all of its game objects, all of our C# scripts were in fact components that could be used and attached to any instance of the GameObject class. Because of this, there were many design thought processes along the way in choosing exactly what a class should contain and how we can best separate them to allow the best possible maintenance and encapsulation for the specific areas of focus.

Examples of this from *Lunch Break* are the CameraMovement and CameraMovementEffects classes. These classes were separated because the CameraMovement class is focused only on the movement of the camera, checking if players are on screen and some slight rotation when the camera moved left to right or over an edge. The CameraMovementEffects class however is concerned only with the effects that occur on the camera. For us that consisted of a camera bounce effect, shake effect and any zooming that needed to be done to keep all players on the screen. Those were all features that I coded and implemented for our game and as such I believe shows a great understanding of game engine concepts relating to ILOs 1 and 2.

We discovered early on that certain functions and actions in the Unity scripts was causing or could cause performance issues. As a team we decided that we should try to avoid using any GetComponent or similar calls every tick/update as they were quite resource intensive. We went through many of our scripts and changed any that had an intensive function call within the Update() method and moved it to the Awake() or Start() methods and stored its value in a class variable. This way we were still able to reference a certain object on every update but did not need to find and retrieve the reference every tick which would just be silly if it’s avoidable. As this was a performance and optimisation based decision I believe it links in perfectly with the performance ILO (3).

As Unity scripts are components within the Unity system, less of the other structures and patterns were used as they either just weren’t necessary and would be overkill or the Unity system aided us enough with its own inbuilt API to not require extra patterns. One of the concepts used often though was the State Machine, as it can be as simple or complex as a scenario needs. For example I used a basic state machine in the CameraMovementEffects class for the bounce effect. This effect would occur when a player took damage or collided with another scene object. The state machine for this was very simple and stripped back as it did not need to be overly complex. The state was stored in an int bounceState object and represented where the camera was in its bounce. When the camera was not bouncing the state would equal 0, but when a bounce started the state would be set to 6 and decrement by 1 each time it completed a portion of its bounce (a LERP to a specific position). Using this basic state machine I was easily able to set up a switch statement to handle the new desired position of the camera after each LERP and modify the intensity of the bounce with each movement. We also used basic state machines like the above in the Player class and main menu, except using enums as they suited that scenario much better than an integer. These examples show and prove my understanding of structures and patterns and when and where to use them in games, relating directly to the maintenance ILO (4).

# What I Could Have Done:

In semester 1 this year, prior to me completing this unit, I knew a lot less about game engines and many of the concepts used in games programming. Early on I worked primarily on the camera follow script with multiple iterations as well as pause menu and main menu UI components and the scripts to allow those menus to be used with controller and keyboard input.

In some of my very early menu scripts I tried many different ideas to get a nice solution to the menu problem. I remember trying something similar to a state machine, where dependant on which menu option was selected it would enable certain areas of code and display certain UI components/colours, but this just got far too messy. It was becoming a giant switch statement or far too many if clauses. I later went back and changed it to an array of UI components, which was then navigated through using the players input. This ended up being a much nicer solution but still possibly not the best one.

I believe had I done this subject earlier I would have been able to design and implement a much nicer solution than both examples above and had much more time to spend on other features of the game during semester 1.

# The Game:









# Code:

Some classes from the game have been included in the submission USB as an added reference and proof of my distinction work and to show off some of the code I did.