# CA 1 – Design Doc

# 3D Game Engine Development

# Luke Courtney

## Overview

This game was created as part of CA 1 for 3D Game Engine Development. The objective of this assessment was to create a game under the theme of *“Start from nothing”*. In it, you control a ball in a dark, maze-like environment. Due to the oppressive darkness and lack of visibility, you must rely on the light from a flashlight to find your way around. From the darkness comes an endless wave of enemies, that when they see you will attempt to chase and kill you. If you get hit three times, its game over. These enemies can be killed using the light from the flashlight. This flashlight however has a limited amount of battery, and drains over time. As you progress through the game, the rate at which it drains also increases. Randomly spawning pickups can be found throughout the level can be found and used to replenish health and flashlight battery. Over time the enemies will appear more frequently, as well as in more powerful forms. Your objective is to survive as long as possible in this environment.

A light in the dark

Description automatically generated

Figure - Game scene at start.

The game takes place in a pitch-black environment designed to be disorienting and maze-like. The darkness in combination with the mix of wide open areas and narrow spaces make it difficult to keep track of where you are and where you have been. This creates a tense atmosphere as you try to figure out the layout of the level, and remember where you’ve been already.

The game features a single mode. Starting as soon as the player hits start on the main menu, the goal is to survive as long as possible. The game starts off slowly, with enemies spawning every 10 seconds randomly across the map. The difficulty soon increases however in a number of ways.

* Enemies spawn more frequently
* Special enemy types spawn
* The players flashlight battery drains faster
* Health packs and batteries appear less often
* As the player is chased by more enemies, the camera zooms in and a faster heartbeat sound can be heard

As these difficulties build, the player also unlocks a number of upgrades that can be used in order to tip the scales in their favour. These include:

* Dash ability
* Area-of-Effect damage ability
* Ping ability that highlights enemies
* Teleport ability that teleports player to a random point on the map

These abilities are unlocked through defeating enemies, starting with the dash ability at 5 kills, and working up to the teleport ability much later in the game. Each ability has a different cooldown. The dash ability can be used every 5 seconds. for example, but the teleport ability takes a total of 5 minutes to recharge after use.

A blurry image of a light

Description automatically generated

Figure - Player with 3 lives and all abilities ready

The game contains minimal on-screen text or UI elements, instead opting to display relevant information to the player in-world. For example, the players current health is displayed as a series of rings of light around the player, and the status of the players abilities are shown by a set of coloured lights behind them.

A screenshot of a video game

Description automatically generated

Figure - Main Menu

When the player launches the game, they will be brought to the main menu. From here they have the options of starting the game, viewing their scores, viewing the tutorial, and quitting the game. The menu can be navigated using the *W* and *S*  keys, and selections can be made using the *Enter* key. The current selection is indicated by the two white spheres with lights. The menu is accompanied by custom menu music made in FL Studio. The same music can be heard when the player is killed.

A screenshot of a video game menu

Description automatically generated

Figure - Scores menu

The game features a scores menu which keeps track of the total number of kills, deaths, health pickups and batteries collected by the player. These are lifetime stats, and are carried over through as times the player plays the game.

A screen shot of a computer

Description automatically generated

Figure - Tutorial Slide 3

From the main menu the player can access a tutorial. This gives the player a brief summary on how to play the game. They are instructed on how to toggle their flashlight on and off to save battery, to collect pickups such as health and batteries, how to unlock and use powerups, and the objective of the game.

## Development

The game contains a total of 21 scripts, each with different functions. The 14 of these scripts are used in the main game mode of the game, while the rest are used in the different menus and tutorial scenes. In Unity, each script is attached to a game object as a component. The script can then act upon the object it is attached to, and the scene around it. In some cases, such as in the case of a player controller script, the scripts primary function is to control the player object it is attached to. A script such as the wave manager script is used to control elements of the scene, in this case the spawning of enemies and pickups. Because the wave manager script does not control a particular game object, it is attached to an “empty” game object.

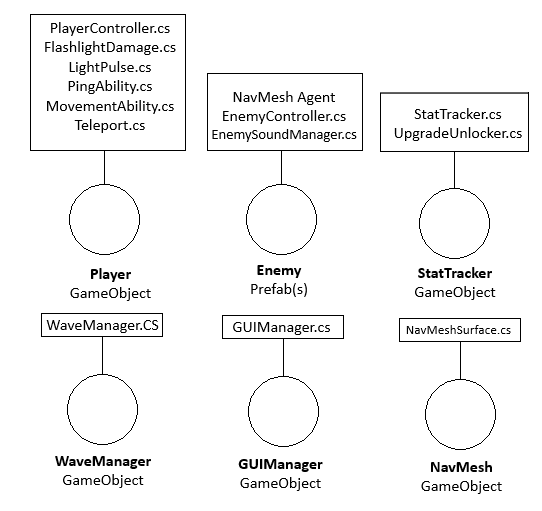


Figure – Game mode GameObject Diagram

Shown above is a diagram showing each script used in the main game mode of the game, and the game object it is attached to. The first object in the diagram, the player game object, has a total of 6 scripts attached to it. These scripts perform a number of functions including controlling the player movement, to allowing the player to damage the enemies with their flashlight, and activating the powerups unlocked over the course of the game.

Similarly, the Enemy prefab uses a number of scripts related to its movement and functions, including a NavMeshAgent script. This script allows the enemy to be moved around in the scene using a NavMeshSurface.

The StatTracker, WaveManager, GUIManager and NavMesh game objects are all “empty” game objects. This is because they are not intended to act on one specific object, but on the scene as a whole. Scripts attached to regular game objects can also act upon other objects in the scene, but empty game objects are more suited to this role as they do not have any model, instead only having a position in the scene.

Each script in the project serves its own purpose. I made an effort to keep different parts of the game in separate scripts. For example, each of the players abilities are in separate scripts. These abilities are unlocked after a number of kills, which is kept track of by the StatTracker script, and the abilities are unlocked with the UpgradeUnlocker script. Due to the number of different scripts used, I will be unable to cover them all here, but I will give an overview of the most important elements of the game.

A computer code on a black background

Description automatically generated

Figure - Flashlight damage DamageBeam() function

The flashlight damage script allows the player to damage and kill enemies using their flashlight. This is done by casting a ray outwards from the player in the direction of the flashlight beam and detecting enemies that collide with it. Any enemy hit by the ray will be dealt a level of damage determined by their distance. The further away the enemy is, the less damage they will take. As the players flashlight battery drains, the max range of the beam decreases.

A computer screen shot of a program

Description automatically generated

Figure - Flashlight damage DepleteBattery() function

Also in the flashlight damage script is the DepleteBattery() function. This function causes the battery to drain over time. The flashlight intensity can then be updated to reflect the new battery level. The rate at which the battery drains increases over time, to a maximum of 2.5. This can be seen from line 82 to 85 in figure 8 above.

A computer screen shot of a program code

Description automatically generated

Figure - Player detection in enemy controller script

Enemies are a vital part of the game. They must be able to detect when a player is close, and if they have line-of-sight to the player before they start chasing them. This is done by first waiting until the player is within the maximum detection range for this enemy type, and then casting a ray to the player position to determine if the enemy is within line-of-sight. If both of these conditions are met, the enemy uses the NavMeshAgent component to set destination to the current player position. There is a 500ms delay between each call of the SetDestination() function for the sake of performance. Once the enemy is on the move, their speed is set to 10.5f, significantly faster than their idle wander speed.

A screen shot of a computer program

Description automatically generated

Figure - StatTracker tracking kill stats

In order to keep track of the player stats, the PlayerPrefs system is used. This allows us to store values that persist even after the game is closed. Shown above in figure 10 is the set of functions used to keep track of any kill stats. Note the use of two get functions, GetKills(), and GetCurrentKills(). When a kill is recorded as a PlayerPref, it is added to the lifetime total number of kills obtained by the player. In order to keep track of stats just for one round in order to display them a the end, we use an integer variable in addition to the PlayerPrefs. The GetCurrentKills() function returns this variable. The variable values are reset at the start of each round.

A computer screen shot of text

Description automatically generated

Figure - Upgrade Unlocker

Another core feature of the game is the ability to unlock upgrades as the game progresses. This is done using the upgrade unlocker script. As shown in Figure 11, the script first takes in the player score, which in this case is the total number of kills that round. It then checks if the player has an ability. If they do not already have it, the current score is compared to the total score required to unlock that ability. If met, the ability is unlocked and a sound effect is played. This process is repeated for each ability.

A screen shot of a computer code

Description automatically generated

Figure - NearestNavmeshPoint() function

An important function found in several scripts is the NearestNavmeshPoint() function. This was created initially to spawn enemies and went on to be used to place health and battery pickups and to teleport the player when they activate the warp ability. The function takes in a point in the form of a Vector3, and finds the nearest point to that point somewhere on the navmesh. This allows us to generate a random point within the map bounds, and reasonably close to the navmesh, and move it to be directly on the navmesh. The function only looks for a point within 5 meters for the sake of performance. Because we know the map size, and the range of elevation, we can make sure to always generate an initial point within 5m of the navmesh. Once a point on the navmesh is found, we can then spawn an enemy or pickup there, or teleport the player to that location. This ensures that any randomly generated point for spawns will not be inside a wall, or any other invalid position. If no nearest point on the navmesh is found, a debug statement is printed, and the initial point is returned.

## Unity Development

Over the course of this project, I gained a significant amount of experience in developing games in Unity. Though we have spent time using Unity in the past, completing the Unity Create with Code course for example, this proved to be the most valuable learning experience with the engine yet. In this project we were free to design a game in our own way, using the basic skills obtained through the Create with Code course, and then learn new ones as needed by the project. This not only helped develop new skills and experience with the Unity editor and programming in C#, but also provided further experience in researching and finding solutions to problems and challenges in the Unity editor and C#.

I personally found the experience of using Unity to be significantly smoother and more enjoyable than that of game development in C++. The value of ability to see changes being made in the scene as they happen cannot be overstated. Additionally, the ability to create separate scripts and attach them as components, as opposed to having single large classes proved to make keeping track of and working with many scripts significantly easier.

Overall, I enjoyed the process of designing and developing a game in Unity. This project was a valuable experience in doing so and served to test not only what we have learned about Unity from the Create with Code course, but also our ability to research and find solutions on our own.