Assignment Submission

Student Name	Tríona Barrow
Student Number	11319851
Degree Programme	B. Sc. in Computer Applications (Software Engineering)
Module Title	Fourth Year Project
Module Code	CA400
Project Title	Quest
Lecturer	Paul Clarke
Supervisor	Renaat Verbruggen
Project Due Date	21st May 2018
Assignment	Technical Guide
GitLab Link	https://gitlab.computing.dcu.ie/barrowt2/2018-ca400-barrowt2

Declaration

0. Table of Contents

1. Abstract

The project discussed in the following guide is a 2D, top-down exploration game. Quest was built using the Unity Game Engine with C# as the programming language, and it is intended for users with an interest in gaming and adventure storylines. The project has an emphasis on adventure and puzzle solving. The player controls a player sprite to complete quests to complete the game.

I wanted to build my own game due to my own interest in gaming, so through my own research I decided on C# as the programming language and Unity as the game engine. Unity allows the use of a number of components to ensure that the right objects are presented on screen, which can then be backed up by the programming language to ensure the correct functionality is being carried out.

2. Motivation

One of my major interests and hobbies is gaming, including traditional card and board games, as well as PC games. Through work experience in VSware and BearingPoint and modules such as Human Computer Interaction - I became more interested in the design and implementation of something like a game. Through my work in BearingPoint, I was more exposed to the C# language, and wanted to take on a project that used this language separately to the web application stack used in work.

With the combination of the interest I had developed in gaming and system design, a game seemed like a good fit for a project to take on. Gaming as a whole has also shown improvements in key skills for users, including concentration, decision making and multitasking. The indie games segment is a steadily growing market, so developing project allowed me to gain valuable insights into game development and widen my skill set.

3. Research

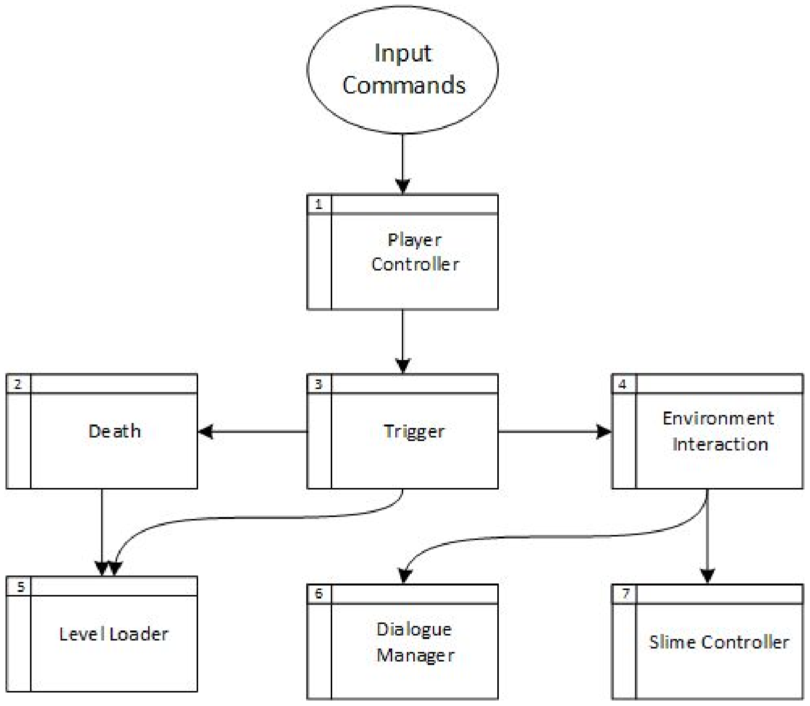
Before embarking on the development of this project, I had a good idea of what was needed in terms of the design and implementation. In general, the research I carried out for this project was around specific areas and components I could incorporate into it. I had previously used the Unity game engine for my third year project in 2015/2016, however the API and editor have gone through significant updates since then. This required time to re-familiarise myself with these, and understand the changes that were made during this time.

I also had just started an intern position at BearingPoint, working on a team developing primarily with C#. This allowed me to gain some valuable insights into using C#, and afforded me the opportunity to become more familiar with the language and its quirks. While some of the standard C# implementation has been modified for the Unity Engine, this familiarity allowed me to debug errors which arose more easily.

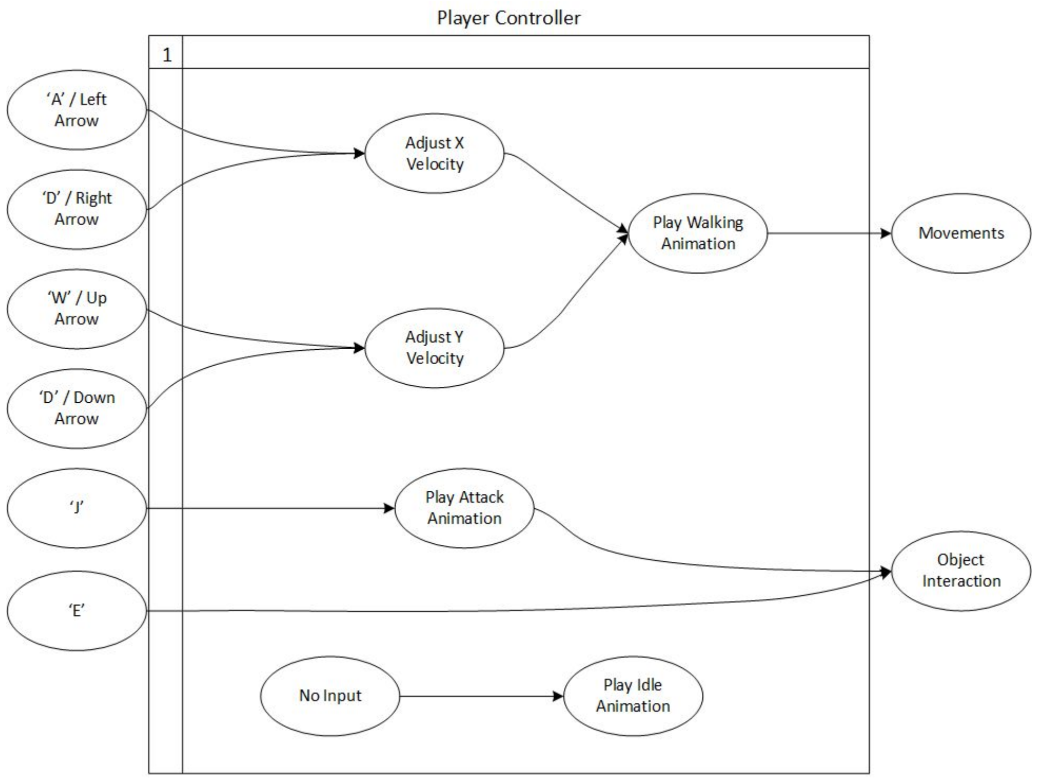
In the previous project I had worked on, I had spent a significant amount of time creating assets for the artwork. This time I wanted to focus more on the implementation and creation of the game, so I researched and found free assets by Kenney (kenney.nl) to use within my game. I also chose free sound effects and music, including music by Kevin MacLeod.

4. Design

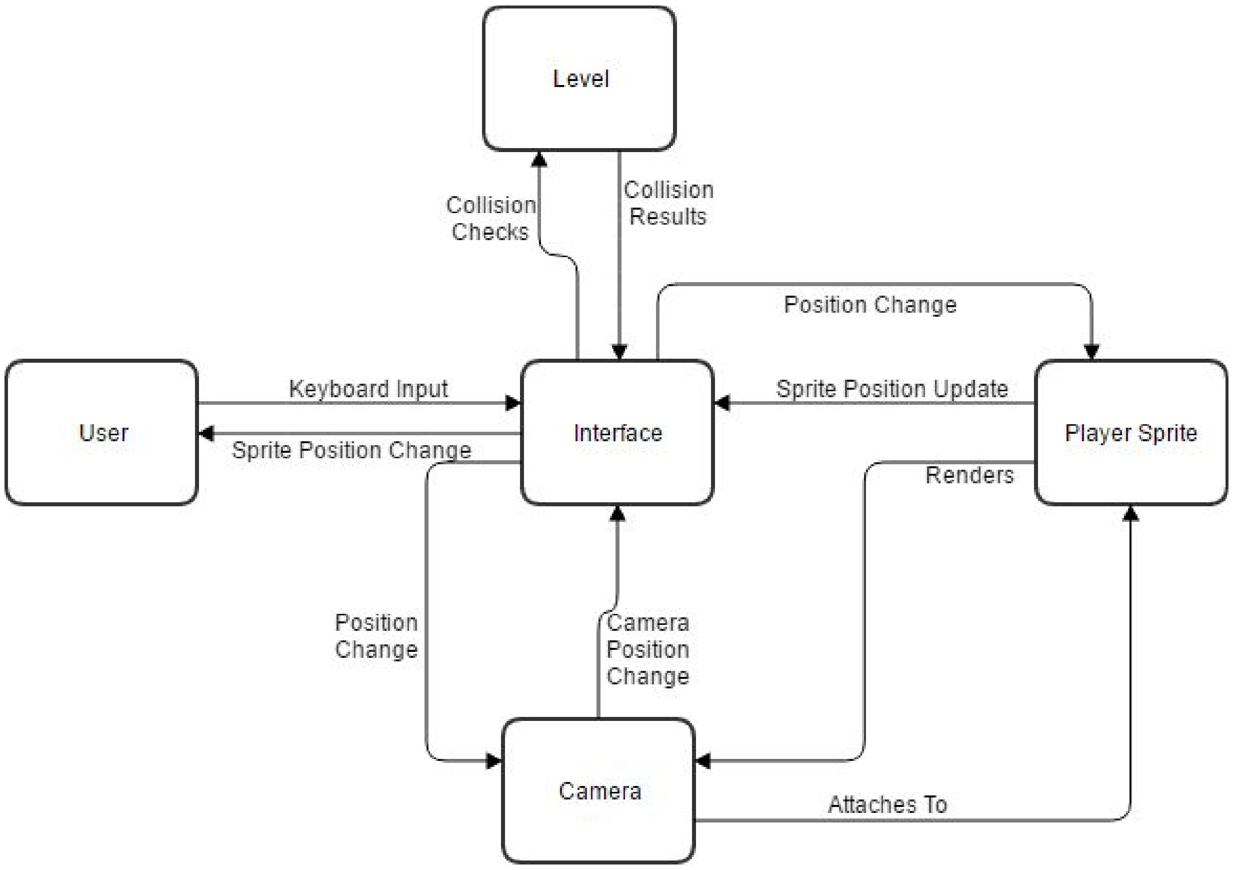
a. Data Flow Diagram - Level 0



b. Data Flow Diagram - Level 1



c. System Context



- *User ←Interface→ Player Sprite*

The User interacts with the Player Sprite by use of the interface within the game. This is carried out via key presses on the keyboard, for example ‘D’/Right Arrow moves the sprite positively along the X axis, while ‘A’/Left Arrow will move it negatively along the same.

- *Player Sprite ←Interface→ Camera*

The Camera is mapped via the CameraController script to follow the Player Sprite with some bobbing for smooth movement across the level. This means that when the position of this sprite is updated across the X and/or Y axis, the camera must also move similarly to ensure consistency.

- *Level ←Interface→ Camera*

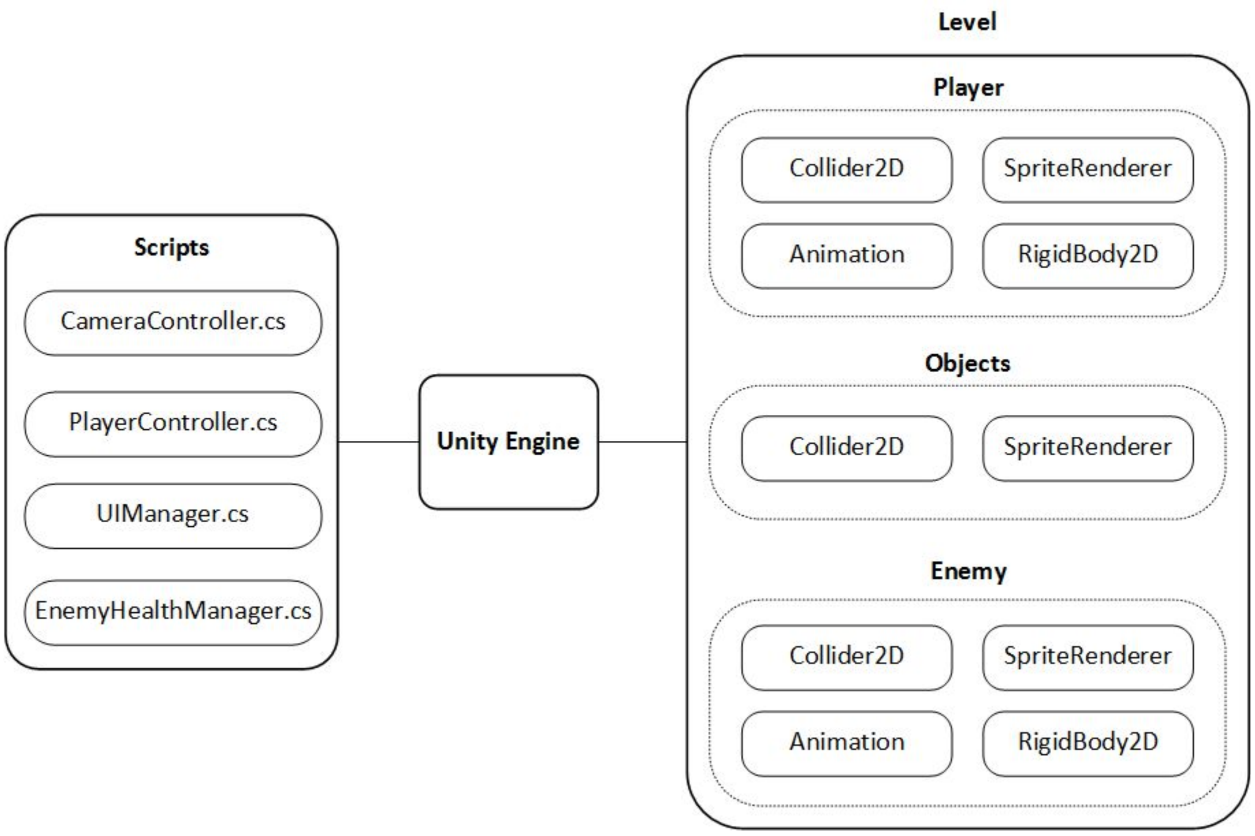
The Camera has bounds set depending on the Level currently being played, so will stop moving past that point along the X/Y axis in the direction the player is moving in - this is set to the level edges.

- *Player Sprite ←Interface→ Level*

The Player Sprite is loaded and rendered into the Level, and moves along other objects contained within the Level. The Player will be able to carry out interactions based on their placement in the Level - if they are within a certain trigger the Player can press ‘E’ to interact with an object in the Level, and press ‘J’ to attack an enemy.

5. Implementation

a. *System Architecture Diagram*



The game consists of a number a scenes - five for the levels the user will progress through, and three for the Start Menu, a Win screen and a Lose screen. Upon opening the game, the user will be presented with the Start Menu - consisting of a number of UI elements. The background is created with a canvas, which the buttons are laid over. On this initial menu, the user has four options. They can click Play to enter the game, and this will load the Forest level, or Options to adjust the volume, Controls to view the controls for the game, or Quit to close the program. If they select Controls, the Menu item is deactivated from view and the Controls are enabled in the view, similarly to the Options menu. On the Options menu, the user is presented with a slider to adjust the volume of the game - this will be saved to the PlayerPrefs for retrieving.

Once the user selects Play, they will be loaded into the Forest level - they will be able to control the Player Sprite in the screen using the controls. The camera will follow the player sprite as it moves around within the level bounds - if it reaches one of these boundaries, then it will stop moving until the player moves away from these bounds again. The player will be able to interact with NPCs and signs in the level by pressing “E”, and can gain information about what the objectives are and where other levels are from this. Dialogue will be displayed at the bottom of the screen, where the player can move through the lines of dialogue using the “Space” key. From this level, the user can progress to three other areas - Town, Cave and Tavern_Outside. In the top right corner of the screen, the player’s current health and a coin counter will be shown visually, so the user can keep track of this.

If the player chooses to proceed to the town, he will load into a small town that has been overrun by enemy slimes. The player must attack these using their sword by pressing “J”. If the player runs into them, their health bar will drop by the amount of damage that enemy deals. If the player kills an enemy, they will drop a coin on the ground. If the player walks over this coin, it will be picked up and the value of the coin will be added to the coin counter in the top left hand corner. The enemies in here will have different speeds and damage to deal, and are visually different by colour. The player can choose to proceed back to the forest level by returning to the bottom right corner, where a trigger will load them back into the level.

If the player proceeds Tavern_Outside from the Forest, they will be loaded into another section of the forest with a building in it. They can proceed into this building by entering the trigger at the doorway, which will load the Tavern_Inside level. Once this loads up, the player will be inside the tavern. The tavern will have NPCs that the player can interact with, and will have a golden bottle on the bar that they can interact with. If they interact with this bottle, their health will be restored to full. The player can retrace their steps from there to return to the forest level.

The final level is the Cave level, where the player will have to fight through a number of enemies. There are multiple types of enemies, who will drop coins during this fight. The player will have to collect all the coins from all the enemies within the game (175 coins) in order to win the game - the game will not load the WinScreen level until that number of coins have been collected.

If the player health reaches zero, then the LoseScreen level will load. This is a canvas with text explaining the user has lost, which remains on screen for five seconds then redirects back to the MainMenu. From here the user can choose to re-enter the game, however their health will go back to full and their coin count will return to zero. If the user successfully beats the game, then they will be met with the WinScreen, another canvas with text explaining the user has won. This also remains on screen for five seconds, before redirecting back to the MainMenu.

6. Sample Code

a. *PlayerController.cs*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour {

      public float attackTime, moveSpeed;
      public Vector2 lastMove;
      public GameObject weapon;
      public bool canMove;

      private Animator anim;
      private bool playerMoving, attacking;
      private Rigidbody2D myRigidBody;
      private static bool playerExists;
      private float attackCounter;
      private SFXManager sfxManager;

      void Start () {
            canMove = true;
            anim = GetComponent<Animator>();
            myRigidBody = GetComponent<Rigidbody2D>();
            sfxManager = FindObjectOfType<SFXManager> ();
            if (!playerExists) {
                  playerExists = true;
                  DontDestroyOnLoad (transform.gameObject);
            } else {
                  Destroy (gameObject);
            }
      }

      void Update () {
            CheckAndEnableSprite ();
            playerMoving = false;

            if (!canMove) {
                  myRigidBody.velocity = Vector2.zero;
                  return;
            }

            if (!attacking) {

                  if (Input.GetAxisRaw ("Horizontal") > 0.5f ||
                    Input.GetAxisRaw ("Horizontal") < -0.5f) {
                myRigidBody.velocity = new Vector2
                    (Input.GetAxisRaw ("Horizontal") * moveSpeed,
                    myRigidBody.velocity.y);
                playerMoving = true;
                lastMove = new Vector2 (Input.GetAxisRaw
                    ("Horizontal"), 0f);
            }
            if (Input.GetAxisRaw ("Vertical") > 0.5f ||
                    Input.GetAxisRaw ("Vertical") < -0.5f) {
                myRigidBody.velocity = new Vector2
                    (myRigidBody.velocity.x, Input.GetAxisRaw
                    ("Vertical") * moveSpeed);
                playerMoving = true;
                lastMove = new Vector2 (0f, Input.GetAxisRaw
                    ("Vertical"));
            }

            if (Input.GetAxisRaw ("Horizontal") < 0.5f &&
                    Input.GetAxisRaw ("Horizontal") > -0.5f) {
                myRigidBody.velocity = new Vector2 (0f,
                    myRigidBody.velocity.y);
            }
            if (Input.GetAxisRaw ("Vertical") < 0.5f &&
                    Input.GetAxisRaw ("Vertical") > -0.5f) {
                myRigidBody.velocity = new Vector2
                    (myRigidBody.velocity.x, 0f);
            }
        }

        if (Input.GetKeyDown (KeyCode.J)) {
            attackCounter = attackTime;
            attacking = true;
            myRigidBody.velocity = Vector2.zero;
            anim.SetBool ("Attack", true);

            sfxManager.attack.Play ();
        }

        if (attackCounter > 0) {
            attackCounter -= Time.deltaTime;
        }
        if (attackCounter <= 0) {
            attacking = false;
            anim.SetBool ("Attack", false);
        }
```

```
		anim.SetFloat ("MoveX", Input.GetAxisRaw ("Horizontal"));
		anim.SetFloat ("MoveY", Input.GetAxisRaw ("Vertical"));
		anim.SetBool ("PlayerMoving", playerMoving);
		anim.SetFloat ("LastMoveX", lastMove.x);
		anim.SetFloat ("LastMoveY", lastMove.y);
	}

	void CheckAndEnableSprite() {
		if(!gameObject.GetComponent<SpriteRenderer> ().enabled)
			gameObject.GetComponent<SpriteRenderer> ().enabled =
				true;
	}
}
```

b. DialogueManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class DialogueManager : MonoBehaviour {

	public GameObject dialogueBox;
	public Text dialogue;
	public bool active;
	public string[] dialogueArray;
	public int currentLine;

	private PlayerController player;

	void Start () {
		player = FindObjectOfType<PlayerController> ();
	}

	void Update () {
		if (!active) {
			dialogueBox.SetActive (false);
		} else if (active && Input.GetKeyDown(KeyCode.Space)) {
			if (currentLine >= dialogueArray.Length - 1) {
				dialogueBox.SetActive (false);
				active = false;
				currentLine = 0;
				player.canMove = true;
			} else {
				currentLine++;
			}
		}
```

```
            dialogue.text = dialogueArray [currentLine];
      }

      public void SetActive() {
            active = true;
            dialogueBox.SetActive (true);
            player.canMove = false;
      }

      public void ShowBox(string dialogueText) {
            SetActive ();
            dialogue.text = dialogueText;
      }

      public void ShowDialogue() {
            SetActive ();
      }
}
```

c. SlimeController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SlimeController : MonoBehaviour {

      public float moveSpeed, timeBetweenMoves, timeToMove,
                  waitToReload;
      public bool reloading;

      private Rigidbody2D myRigidbody;
      private bool moving;
      private float betweenMoveCounter, toMoveCounter;
      private Vector3 moveDirection;
      private GameObject player;

      void Start () {
            myRigidbody = GetComponent<Rigidbody2D> ();
            SetBetweenMove();
            SetToMove();
      }

      void Update () {
            if (moving) {
                  toMoveCounter -= Time.deltaTime;
                  myRigidbody.velocity = moveDirection;
            if (toMoveCounter < 0f) {
                  moving = !moving;
                  SetBetweenMove();
            }
      } else {
            betweenMoveCounter -= Time.deltaTime;
            myRigidbody.velocity = Vector2.zero;
            if (betweenMoveCounter < 0f) {
                  moving = !moving;
                  SetToMove();
                  // Choose a random direction
                  moveDirection = new Vector3 (Random.Range (-1f,
                        1f) * moveSpeed, Random.Range (-1f, 1f) *
                        moveSpeed);
            }
      }

      if (reloading) {
            waitToReload -= Time.deltaTime;
            if (waitToReload < 0) {
                  reloading = false;
                  SceneManager.LoadSceneAsync
                        (SceneManager.GetActiveScene().buildIndex);
                  player.SetActive (true);
            }
      }
   }

   void SetToMove() {
      toMoveCounter = Random.Range(timeToMove * 0.75f, timeToMove
             * 1.25f);
   }

   void SetBetweenMove() {
      betweenMoveCounter = Random.Range(timeBetweenMoves * 0.75f,
             timeBetweenMoves * 1.25f);
   }
}
```

7. Problems Solved

a. *Animation*

Animation was one of the areas of Unity that had changed significantly from the last time I used Unity. Previously, you could set up an image per frame of the Animation window in Unity and it would combine these into an animation for you to use. However, it has become more complicated - to set each frame you must hit record while arranging all the objects. If you do not do this, then between frames Unity will try to correct the objects back to their original rotation and position, leading to an odd “drifting” appearance for some of these objects. It was an easy fix once I realised, but it took a long time to find the root of this issue.

b. *Respawning Objects*

Multiple objects within the game, such as dialogue areas and the player sprite, required swapping between active and inactive within the level. However, I ran into an issue by setting these objects as inactive using `SetActive(false)` - this would completely disable the object, meaning you could not call it directly to set it as active again. To work around this, I either disabled the SpriteRenderer component of the object or had it as a child of another object, so the parent could re-enable it.

c. *Time*

While I didn’t have any other university commitments, I was working throughout the academic year within the industry as a full time developer. This impacted on the time I had available to me to dedicate to the project, both in regards to the normal working week and any extra time used for work related events. If I revisited a project like this in future, I would factor this into my planning and ensure that I spend more time earlier on getting the essentials in place.

8. Results

Validating this project was challenging, due to the nature of the Unity game engine. As a live game engine, you cannot carry out code unit testing with C# testing frameworks, such as Xunit. While Unity has started compiling a TestRunner specifically for Unity games, it is quite limited and would not produce the test coverage I would look for from code-based testing.

With this in mind, user testing and manual testing became a key part of the validation of the project. When adding new components and code to the project, I conducted manual testing and used Debug statements to get output in the Console, to ensure that the functionality I needed was being fulfilled. The Unity console also outputs general exceptions, such as NullReferenceException and IndexOutOfBoundsException, which can be useful to ensure that the correct behaviour is being carried out.

My main concern was ensuring that I would have a functioning project by the submission date, and I feel this has been achieved. While there is further work and improvements that could be carried out on it, there is a logical start, plotline and end of the game. I also gained valuable experience working on a project like this alone, as many college assignments and work tasks were carried out within a team. Overall I am happy with the features that I implemented, and the skills I honed throughout the process.

9. Future Work

a. More Levels

The current version of the game is quite short, with only a few levels and objectives for the player to complete. With more levels and more quests, I feel like this would add to the complexity of the game, and help to keep the player's interest in this.

b. Multiplayer

As part of my functional specification, I highlighted that the option for multiplayer would be great for this game. While I wasn't able to deliver on this within the time frame, I feel this would be a positive change to the project as it would expand the ways that users can interact with the game.

c. AI Improvements

With more levels and a multiplayer aspect, I feel improvements to the AI for the enemies in the game would be beneficial for the user experience. More intelligent enemies can increase the perceived challenge in the game and increase the difficulty, appealing to players who look for more complex challenges.

d. Save/Load Game

While the current version of the game is quite short - if the number of levels and events in the game were expanded I feel the user would need a save/load game function. This would mean that the user is not losing progress from having to leave the game, and can come back it at a time that suits them better.