Assignment Submission

Student Names:	Kieran McEvoy Tríona Barrow
Student Numbers:	12362641 11319851
Degree Programme:	B. Sc. in Computer Applications (Software Engineering)
Project Title:	Third Year Project
Module Code:	CA326
Lecturer:	Charles Daly
Supervisor:	Alistair Sutherland
Project Due Date:	18th March 2016
Group Number:	46
Assignment:	Technical Specification

Declaration

We declare that this material, which we now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. We have read and understood the Assignment Regulations set out in the module documentation. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.

ALL ALONE?

Table of Contents

1. Introduction

1.1 Overview

‘All Alone?’ is a side-scrolling exploration game, with an emphasis on problem solving. It has been created using the Unity 5 Engine in 2D, and using C# as the backing language (used in Unity as a scripting language) to add usability to the objects within the game. It is targeted towards the gaming market, which although is extremely competitive and crowded, is a vibrant and growing environment to work in.

The Unity Engine uses a number of system calls and their own API with the language to make objects work - ones that we found were used commonly were RigidBody2D and Collider2D. Both of these are generally used in Unity projects to handle gravity, as well as to ensure that triggers and level objects are being utilised appropriately. The images we created for use in the game have been imported from files within the project folder (similar to the scripts), and are used within Unity with components added to them - such as specific scripts, colliders, UI elements, etc.

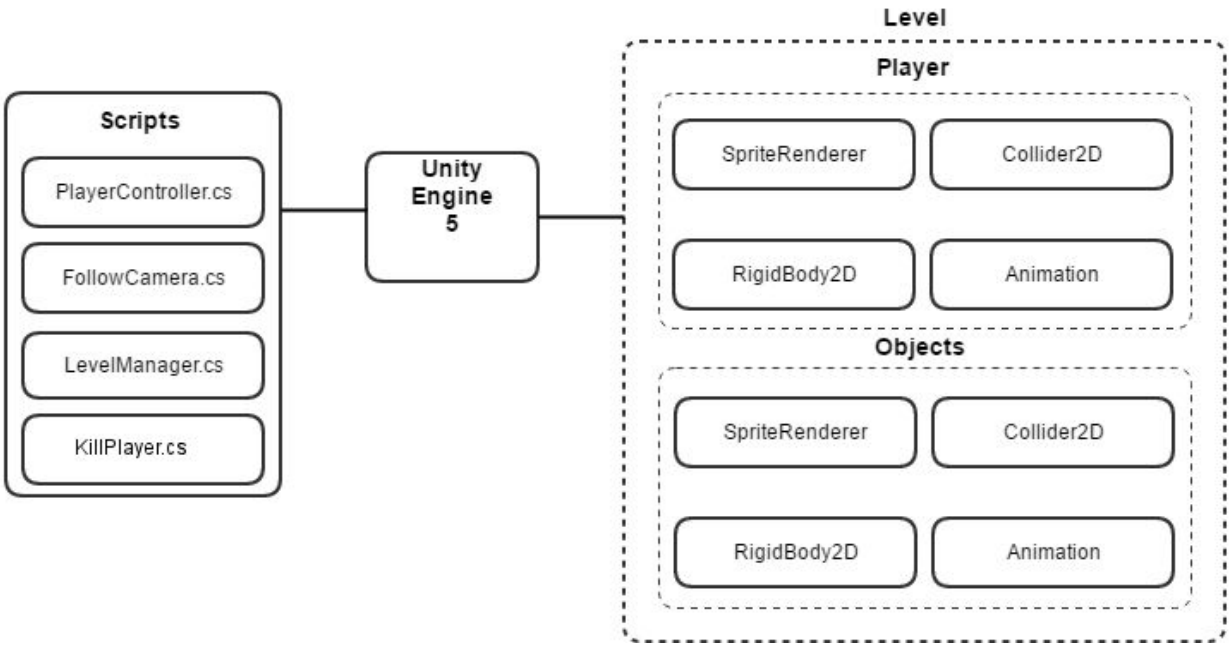
As a built game from Unity, the game itself functions as a .exe file - which means it will be compatible with Windows systems. As we personally use 64 bit Windows, it has been built for this, however it is easy to port to other desktop platforms from the Build settings in Unity. Because of building the game through Unity, it also will match the general system requirements for running a Unity game - in general, it will run on any Windows system past XP Service Pack 2.

1.2 Glossary

Component	Base class for everything attached to GameObjects, including the functions for these GameObjects. This includes Scripts and Unity components, like Audio and UI.
GameObject	An object within the Unity environment that is utilised within the level. Without Components attached, they are essentially empty containers within the scene.
Scripts	Coding files that are used as Components with GameObjects to carry out specific actions within the scene. These can handle a variety of actions, such as user input, changing animations, and more.
Scene	This is a Unity file that contains GameObjects. It can be used for the levels, as well as menus. The build uses Scenes in a specific order for the level order on user playthrough.
Build	When a game has been completed in Unity, it has to have the Scenes required for the game imported into it. It will output files suitable for the platform selected, such as .exe for Windows.
Prefab	A stored copy of a GameObject, with Components attached, that can be duplicated with these attached to every instance of the GameObject that originates from the Prefab.

2. System Architecture

2.1 System Architecture Diagram

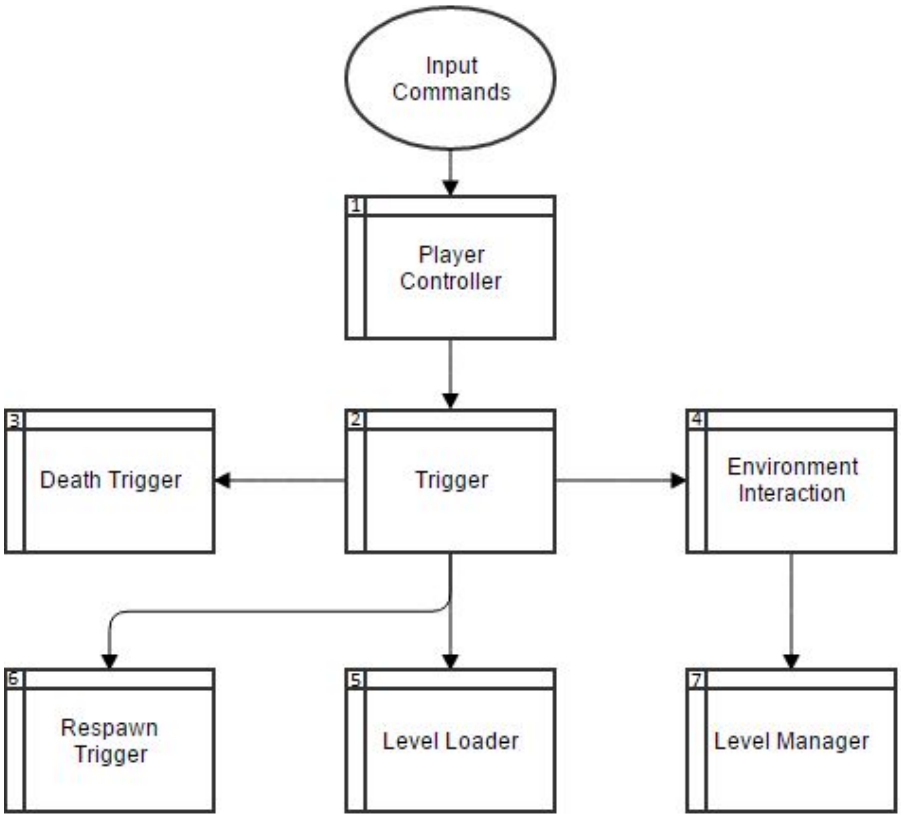


2.1 System Architecture Diagram

The diagram above shows how Unity works, the scripts on disk are imported into the Unity Engine itself, which then interacts with the Level within the game - which is essentially the parent of the Player and the level Objects. Some examples of the scripts we used are details above - such as PlayerController.cs for Player movements, FollowCamera.cs for the camera tracking, and so on. Objects within the game generally use the scripts and components from within the Unity Engine - such as SpriteRenderer for the sprite artwork, Collider2D for collision detection, RigidBody2D for gravity and mass, and Animation for changing the artwork dependant on movements within the game.

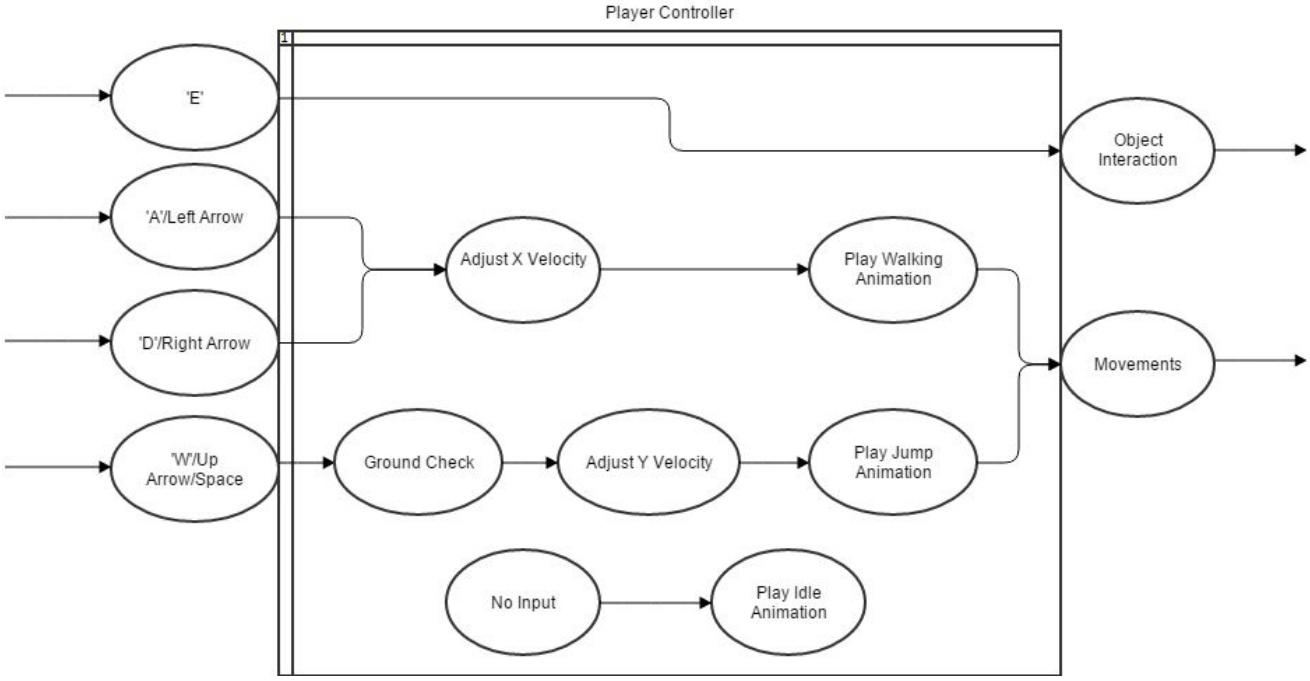
3. High-Level Design

3.1 Data Flow Diagram - Level 0



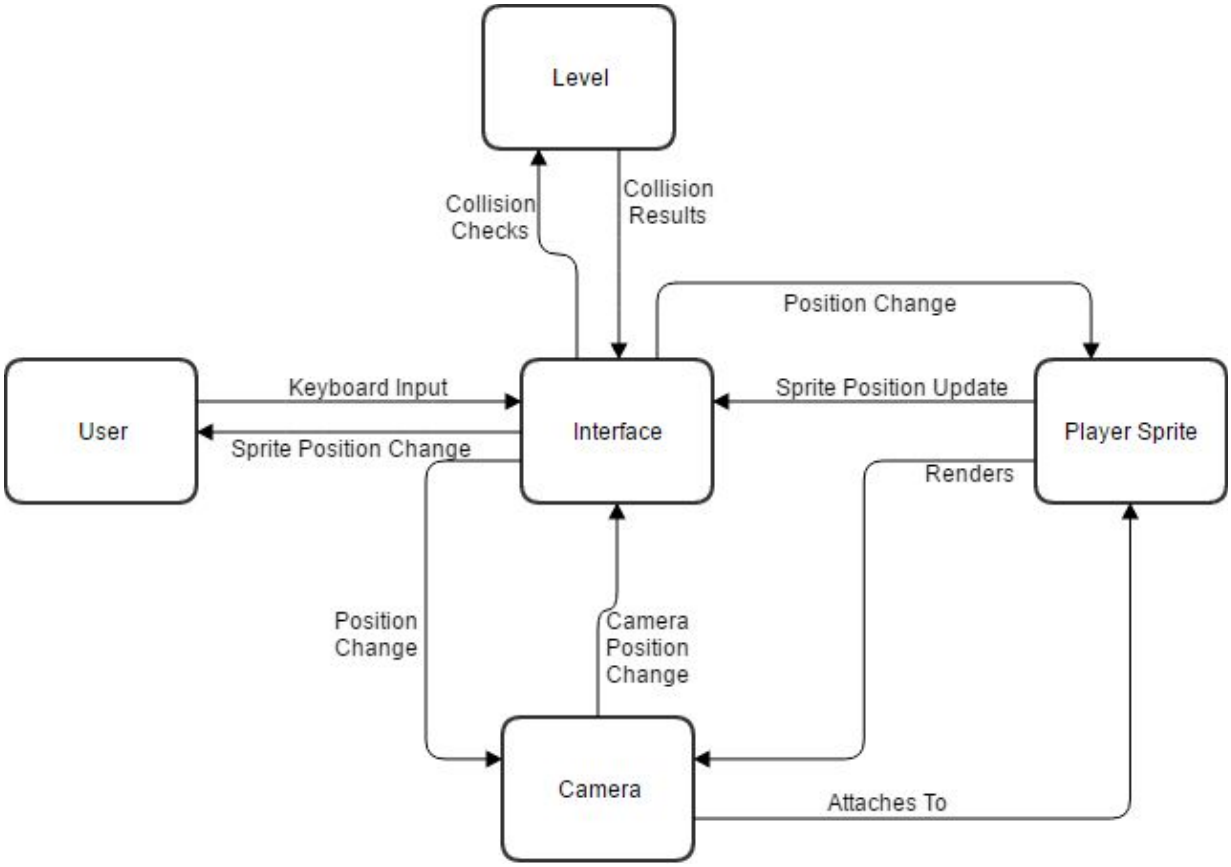
3.1 DFD Level 0

3.2 Data Flow Diagram - Level 1



3.2 DFD Level 1 - Player Controller

3.3 System Context Diagram



- *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, such as ‘D’/Right Arrow moves it positively along the X axis, while ‘A’/Left Arrow will move it negatively along the same. The Vehicle will then respond by increasing the x,y and z coordinates along its current trajectory.

- *Player Sprite ←Interface→ Camera*

The Camera is mapped via the FollowCamera script to follow the player with some bobbing for smooth movement across the level. This means that when the Player Sprite position 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. If a Player walks through a collider set as a trigger, the camera will also zoom out to show more of the level, and revert back to normal once the Player has exited this collider.

- *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 Sprite works with colliders on other objects within Level for checking if movement can be carried out along the X/Y axis. The Player will also 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 modify an object in the Level, and the Player must be on the Ground layer within the level to be able to jump.

4. Problems and Resolutions

4.1 Problem 1 - Sound Playing

In order to ensure an immersive experience for our users, we needed to add sounds as the player’s avatar moved and interacted throughout the levels. The first thing we attempted was adding ambient sounds to the levels itself, to give the idea that the player was in a cave. Our following task was adding sounds to specific events, which proved to be more complex. We initially had sounds coming from different sources, but they were looping, skipping or just not playing at the right time. Through more testing and rewriting of our code, we were able to grasp a good understanding of how to implement sound effectively. Once we achieved this, replicating the process proved much easier.

4.2 Problem 2 - Animation

Part of the animation process within Unity is backed up by scripting, for the Player Sprite we had all of this contained within the PlayerController.cs file. Most of these relied on boolean values - such as isGrounded, isClimbing, and stopped. However, with animation, if there is any point where you are trying to play an animation - you have to ensure that the transitions rely on all the applicable variables, so if you have left out a boolean such as the movement speed, then this can cause the wrong animations to show. By including the relevant booleans we were able to ensure this was all working correctly, however it took time to test the conditions that were affecting this, and update the code to account for this.

4.3 Problem 3 - Object Movement

An important part of the level is the puzzle aspect, which allows the player to interact with the environment to solve the puzzle to allow them to progress further within the level. However, this ended up being one of the harder parts to implement - especially with the bridge movement and light panels. Our initial draft of the bridge movement ended up with the bridge moving negatively across the Y and Z axis, meaning is was moving backwards in the field of view and off the level. The light panels became an issue due to the difficulty within code to get the correct SpriteRenderer components within a parent SpriteRenderer, and ensure the right ones were being enabled and disabled by the button press. This was resolved by using the Unity editor interface to drag in the specific SpriteRenderer objects, and the script required multiple breakdowns and revisions to ensure it was working correctly.

4.4 Version Control

As Unity doesn't include any built in backup features online - this posed somewhat of an issue, as we both used different machines to work on the project together. GitHub became inaccessible with this project also, as we had multiple images, objects, prefabs and scripts to keep together in a specific file order within a filesystem. In the end, we settled on a shared Google Drive folder, and uploaded zipped versions of the project files as a workaround.

4.4 User Testing

We tried to incorporate user testing as much as possible throughout the entire process. By getting our peers, both friends and others, to test out certain actions within the game during the development stage, this allowed us to highlight small issues that needed to be fixed. Some of these including the player getting stuck on ground colliders, not setting certain colliders as triggers, the ability to fall off levels and continue to fall infinitely and other small items. We were able to resolve most of these - however, if we had left more time aside earlier for this, we would have been better able to target these issues more quickly. This is something we will be focusing on in future projects.

5. Installation Guide

5.1 Download and Installation

This game runs as a Windows .exe and is built as an x86 64 bit program. The game files can be found here in a zipped folder: http://bit.ly/258GsuP

1. Download the zipped file from the link and save to a local folder on your computer.
2. Locate the zipped folder within Windows Explorer and extract the files within - they will be named "AllAlone.exe" and "AllAlone_Data".
3. "AllAlone.exe" and the "AllAlone_Data" folder must be in the same folder/directory in your filesystem.
4. Double click on the file, "AllAlone.exe" to launch the program.
5. Your system will launch the game.