GOLDSMITHS, UNIVERSITY OF LONDON

FINAL YEAR PROJECT

'Walkies'; An Infinite Runner-style Video Game made with Unity Game Engine

---

Author:
Molly MASON
33460899

Supervisor:
Frederic Fol LEYMARIE

A thesis submitted in fulfillment of the requirements for BSc Computer Science Degree

May 17, 2019

Abstract

Video games are among the most popular form of media nowadays; accessible on the majority of technical devices and appealing to people of all backgrounds. As a result of this popularity, there is no lack of available tools, tutorials, and software requiring as little as no technical ability; allowing for an explosion of independent games to be made and shared with the world. In this paper, the project 'Walkies', an infinite runner video game made using Unity game engine for desktop devices, is discussed and explored from conception to final release. 'Walkies' takes place in a quaint suburban town, and features AI agent implementation, alongside a variety of infinite runner-style levels that implement multiple techniques, such as random generation spawning.

Acknowledgements

I would primarily like to thank my supervisor Frederic Fol Leymarie for helping guide me through the process, helping me stay on track as well as providing valuable help and feedback as I planned and implemented my project.

Additionally, I would like to express my gratitude to my family and friends who took their time to playtest my project, and provide me with a ton of varied feedback through both survey and conversation form. A big thank you to my friends Kelsey and Jonathan, whom playtested different versions of the game repetitively, determined to find any and every bug, and break the game in every possible way!

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Aims . . . 1
- 1.3 Report Structure . . . 3

2 Background **4**
- 2.1 Gameplay . . . 4
  - 2.1.1 Genre . . . 4
  - 2.1.2 Platform . . . 5
  - 2.1.3 Gameplay . . . 6
  - 2.1.4 Balancing . . . 7
- 2.2 Quality . . . 8
  - 2.2.1 Graphics . . . 8
  - 2.2.2 Bugs . . . 9
- 2.3 Business . . . 10
  - 2.3.1 Monetisation . . . 10
  - 2.3.2 Marketing . . . 11
- 2.4 Post-Release . . . 11
- 2.5 Conclusion . . . 12

3 Specifications **13**
- 3.1 Gameplay Requirements . . . 13
- 3.2 AI Requirements . . . 14
- 3.3 UI Requirements . . . 15
- 3.4 Graphical Requirements . . . 16

4 Design and Implementation **17**
- 4.1 Player Controls . . . 18
  - 4.1.1 Player Movement . . . 19
  - 4.1.2 Other Controls . . . 21
    - Pause . . . 21
    - View Mode . . . 21
  - 4.1.3 Player Interaction . . . 22
    - Level Loading . . . 22
    - Dog Companion . . . 23
- 4.2 AI . . . 26
  - 4.2.1 Spawning . . . 26
  - 4.2.2 Behaviour . . . 27
    - Choose Action . . . 27

Idling, Walking, Running . . . 28
Flocking . . . 29
Destination Reset . . . 30
4.3 Levels . . . 31
4.3.1 Level Loading . . . 31
4.3.2 Road Movement . . . 32
4.3.3 Obstacles and Power-ups . . . 33
Spawning . . . 33
Behaviour . . . 34
4.4 UI . . . 36
4.4.1 Start Interface . . . 36
4.4.2 Pause Interface . . . 37
4.4.3 Bulletin Board Interface . . . 37
4.4.4 Level Over Interface . . . 38
4.4.5 Level Overlay . . . 38
4.5 Graphics . . . 39
4.5.1 Environment . . . 39
Models . . . 39
Hub . . . 40
Levels . . . 40
4.5.2 Characters . . . 40
Models . . . 40
Animation . . . 41
4.6 Scrapped Features . . . 41
4.6.1 Saving . . . 42
4.6.2 Reward System . . . 42
4.6.3 AI Responses . . . 42

5 User Guide **44**
5.1 Technical Specifications . . . 44
5.2 User Controls . . . 44
5.3 Gameplay . . . 45
5.3.1 Hub Gameplay . . . 45
5.3.2 Level Gameplay . . . 46
5.4 Interface . . . 47
5.4.1 Start Interface . . . 47
5.4.2 Pause Interface . . . 48
5.4.3 Bulletin Interface . . . 48
5.4.4 Level Over Interface . . . 50

6 Testing **51**
6.1 White-box Testing . . . 51
6.2 Black-box Testing . . . 53
6.2.1 Conclusion . . . 53

7 Evaluation **54**
7.1 User Evaluation . . . 54
7.1.1 User Feedback . . . 54
7.1.2 Improvements . . . 56
7.2 Specification Evaluation . . . 57
7.2.1 Gameplay requirements . . . 57

Enjoyable to play? 57
Ability to explore hub scene, with appropriate collisions and interactions? 57
Ability to access infinite runner levels from various points in the hub scene? 57
Infinite runner levels with moving roads, randomly spawned obstacles or power-ups, life system, and score system? 57
Appropriate balancing - in terms of level difficulty, overall player control sensitivity/speed, and AI action weighting? . 57
Reward system - hat shop, ability to purchase hats for stars? . . 57
7.2.2 AI requirements 58
Ability to perform a variety of actions: idling, walking, running,and flocking? 58
Ability to perform actions in a weighted fashion? 58
Ability to gain a dog that follows the player upon interacting with the dog AI? 58
Ability to express discontent upon collision with the player? . . 58
7.2.3 UI requirements 58
Overall consistent, thematic, and stylistic design? 58
Start menu - capable of starting or quitting the game? 58
Pause menu - capable of pausing game from hub or levels, and quitting the game? 58
Bulletin menu - contains details about controls and gameplay to inform the player? 58
Level UI - capable of displaying updated lives and score 59
7.2.4 Graphical requirements 59
Overall consistent, thematic, and stylistic design? 59
Appropriate sizing and placement of models? 59
Believable walk, run, and idle animation cycles? 59
7.3 Conclusion 59

8 Conclusion **60**
8.1 Overview 60
8.2 Future Work 60
8.3 Final Comments 61

Bibliography **62**

A Walkies Code **66**
A.1 AIBehaviour.cs 66
A.2 AIDogBehaviour.cs 72
A.3 AISpawn.cs 77
A.4 BulletinUI.cs 78
A.5 LevelLoad.cs 81
A.6 LevelOverMenu.cs 83
A.7 LevelOverUIText.cs 85
A.8 LevelPauseMenu.cs 85
A.9 LevelPlayerController.cs 87
A.10 LevelUIText.cs 90
A.11 MoveFloor.cs 90
A.12 PauseMenu.cs 94

A.13 PlayerController.cs 96
A.14 SpawnBehaviour.cs 103
A.15 StartMenu.cs 105

B Walkies Assets 107

C Walkies Survey 117

D Weekly Logs 121
D.1 Week 1 – w/c 21/01/2019 121
D.2 Week 2 – w/c 28/01/2019 121
D.3 Week 3 – w/c 4/02/2019 122
D.4 Week 4 – w/c 11/02/2019 122
D.5 Week 5 – w/c 18/02/2019 123
D.6 Week 6 – w/c 25/02/2019 123
D.7 Week 7 – w/c 4/03/2019 123
D.8 Week 8 – w/c 11/03/2019 124
D.9 Week 9 – w/c 18/03/2019 124
D.10 Week 10 – w/c 25/03/2019 125

E Original Proposal 126
E.1 Introduction 126
E.2 Methods/Skills 126
E.3 Project Evaluating 126
E.4 Project Planning 127
E.5 References 127

F Design Document 128
F.1 Overview 128
F.2 Controls 129
F.3 Levels 129
F.4 Environment 130
F.5 GUI 131
F.6 Other 132
F.6.1 Hat shop 132
F.6.2 Day/Night system 132
F.6.3 Weather system 132
F.6.4 Loading screen 133

G Preliminary Project Report 134
G.1 Introduction 134
G.2 Aims and Objectives 134
G.2.1 Controls 134
G.2.2 Game levels 135
G.2.3 Game Environment 135
G.2.4 GUI 136
G.2.5 Other 136
G.2.6 Summary 137
G.3 Methods 137
G.4 Project Plan 137
G.5 Progress to Date 138
G.6 Planned Work 139

G.7 Appendix 139
G.7.1 Appendix A 139
G.8 References 141

List of Figures

2.1 Super Mario Galaxy 6
2.2 Candy Crush 8
2.3 Detroit: Become Human 9
2.4 Assassin's Creed: Unity 10
2.5 Goat Simulator 10

4.1 Scene Flowchart 18
4.2 Control Code 20
4.3 Animator 20
4.4 Pause Code 21
4.5 View Mode Toggle Code 22
4.6 Level Load Code 23
4.7 Dog Companion Code 1 24
4.8 Dog Companion Code 2 25
4.9 AI Spawn Code 26
4.10 AI Flowchart 27
4.11 AI Action Code 28
4.12 NavMesh Surface 29
4.13 AI Flock Code 30
4.14 AI Reset Code 30
4.15 Level Load Switch Code 31
4.16 Level Load Difficulty Code 32
4.17 Road Movement Diagram 33
4.18 Road Chain Code 33
4.19 Road Spawn Code 34
4.20 Collision Switch Code 35
4.21 Spawn 'Killbox' 35
4.22 Start Button Code 36
4.23 Quit Button Code 37
4.24 Bulletin Flowchart 37
4.25 Level Over Star Code 38
4.26 Level Overlay Code 39
4.27 Collider 39
4.28 Level Set Dressing 40
4.29 Character Models 41
4.30 Cycle Reference 41
4.31 Player Baseball Cap 42

5.1 User Guide Controls 44
5.2 User Guide Bulletin 45
5.3 User Guide Level Indicators 45
5.4 User Guide AI 46
5.5 User Guide Level Gameplay 46

5.6 User Guide Start Menu 47
5.7 User Guide Pause Menus 48
5.8 User Guide Bulletin Home 48
5.9 User Guide Bulletin About 49
5.10 User Guide Bulletin Controls 49
5.11 User Guide Level Over Menu 50

7.1 Survey Results 1 54
7.2 Survey Results 2 55
7.3 Survey Results 3 56

List of Tables

3.1 Gameplay requirements 13
3.2 AI requirements 14
3.3 UI requirements 15
3.4 Graphical requirements 16

4.1 Object spawns 34

6.1 Test Case 1: Player Controls 51
6.2 Test Case 2: Level Load (for all levels) 52
6.3 Test Case 3: Road Moving 52
6.4 Test Case 4: Spawn Collisions 52
6.5 Test Case 5: Dog Companion 52
6.6 Test Case 1: Graphical Requirements 53
6.7 Test Case 2: Appropriate Balancing 53
6.8 Test Case 3: Intuitive UI 53

List of Abbreviations

AI	**A**rtificial **I**ntelligence
MTX	**M**icro **T**ransactions
RPG	**R**ole **P**laying **G**ame
VR	**V**irtual **R**eality
UI	**U**ser **I**nterface

Chapter 1

Introduction

1.1 Motivation

From puzzle to shooter, adventure to platform, mobile to virtual reality, video games have become an increasingly prominent media form throughout the modern world. The possibilities within game development are endless; countless genres to choose or mix and match from, various forms of device to play on, and numerous target audiences to aim at. This is why my project covers the theme of gaming, with myself delving into the realm of game development; encompassing a variety of skills, primarily technical, however also exploring the graphical side of creation.

Thus, deciding to make a video game provides me the freedom to implement anything I want, whilst improving existing technical skills; i.e. programming, and learning new skills along the way; such as 3D modelling, and animation. Additionally, video games are inherently designed to be a fun form of entertainment; who doesn't like entertainment?

As previously touched upon, game development contains endless possibilities; specifically why it peaks my interest, as it allows developers complete creative freedom to do as they wish. With no 'golden formula' of how to create a successful video game, this also lessens restrictions on what type of video game should be made, and how it should be gone about; the decision can be left up to the developer to decide. With such a huge pool of target audiences to focus on, it is likely that any game will be found appealing by at least someone; while similarities can be noticed between commercially successful games, there is no guarantee or predictability of a games' success until release - and even then, games can blow up years after their initial release.

Another key point that draws me to game development is that little to no specialised skills are required; be it a technical or graphical-based skillset. For example, one can make a pixel-style RPG purely using GameMaker Studio and Paint.net, with limited skills prior. This is one reason pixel art is so popular among indie and independent developers - anyone can make it, or at least easily learn how to from the endless tutorials available across the web. In addition to this, masses of free high-quality assets can be found across the internet for use in various game genres and engines; meaning you don't need to be any sort of professional artist in order to create an aesthetically pleasing game.

1.2 Aims

My aim of this project is to develop a 3D game of the infinite runner genre - some examples of other games within this genre being Subway Surfers, and Temple Run. The infinite runner genre is arguably aimed at a more casual, and wider audience;

this, at least in part, is due to the widespread availability of such games, with them generally being available on mobile devices, thus players being able to dip in and out of them with ease. Infinite runners provide simple and straightforward, yet addictive gameplay; you can't pause the game and continue at a later point, you have to continue in an attempt to beat your high score - avoiding obstacles and collecting power-ups for as long as possible.

Alongside the infinite runner aspect of my game, there will be a open hub world, allowing players free roam to do the levels as they wish; non linearly. The theme of the whole game will be based around dog walking - and thus the name 'Walkies' comes into play, with the term typically used to inform one's dog they are going on a walk. Appealing to a younger audience, the art style will follow that of bright, cheerful, cartoonish and colourful.

Lastly, I aim to introduce an aspect of Artificial Intelligence (AI) into the game. This aspect will involve both dog and human AIs wandering around the open hub world, providing a form of 'life' to the game. These AIs will be somewhat basic; able to perform a few actions, but not being a primary focus of gameplay. Further specific aims of the project are to produce a good-quality, fully-working game for desktop, with the vast majority of graphical assets produced by myself; allowing me full control over the visual aspect of the game.

Technically, implementing such a project will primarily involve knowledge of the Unity3D game engine; my software environment of choice for development. It will also require knowledge of C# programming language, and Blender software - again my software of choice for graphical modelling and animation development. In addition to technical skills, I must be aware of features and common practices of game development - such as UI design, AI design, environment design - and how infinite runners specifically work.

Already proficient in Java, I will be required to learn C# - which shouldn't prove to be too much of a challenge due to their similarities. I expect the main hurdles throughout development to be broadening my knowledge of Unity, and learning how to model and animate in Blender; having only made use of Unity briefly in the past, and having never touched Blender.

My experience of having played multitudes of video games for at least half of my life provides me with at least some familiarity, expectations, and understanding of the features and common practices used within game development. Brief generic research into the infinite runner genre has already enlightened me to the illusionary method used in many infinite runners; the player themselves is not moving, but the floor is - either methodically or randomly spawning currency, power-ups, and obstacles along the way.

By the end of this project I thus aim to learn at least the basic process of game development, C#, basic AI development, how to animate and model in Blender, and how to use Unity in more depth; such variety, complexity, and creativity in content providing a alluring pull to this project.

1.3 Report Structure

Chapter 2: Background
This chapter covers the potential problematic aspects to be considered when making a game. It discusses a wide range of game development considerations that may need to be taken into account when making this project, as well as providing numerous examples of games that are especially known for overcoming any problems within these aspects.

Chapter 3: Specifications
This chapter specifies the various requirements for the project - in terms of gameplay, AI, interfaces, and graphical needs - that should be met in order for the project to be seemed successful. Additionally, each requirement is provided with justification for their inclusion.

Chapter 4: Design and Implementation
This chapter delves into the design and implementation of each aspect of the project. It goes through each section individually - controls, AI, levels, UI, graphics - documenting the process from inception to final version. Design decisions are accounted for along the way.

Chapter 5: User Guide
This chapter relies mainly on visuals to provide a guide to the user on how to play the game. It thus features various labelled screenshots of the game in which to do so.

Chapter 6: Testing
This chapter reports the testing that was carried out throughout the project - both white and black box. Each test case is is noted as fulfilling or failing to meet its requirements.

Chapter 7: Evaluation
This chapter evaluates in detail each aspect of the project. User feedback and survey results from playtesting are discussed, as well as the specifications from Chapter 3 - whether they have been met, and can be considered a success, and why.

Chapter 8: Conclusion
This chapter concludes the project, reflecting on the process and final product. It also discusses possible future work that could occur; improvements and additional features that could be added to the game. The report is wrapped up with some final thoughts.

Chapter 2

Background

2.1 Gameplay

Creating a video game doesn't provide a typically obvious 'problem' to solve; instead, game development presents a variety of different problems, some of which I will cover throughout this chapter.

Due to the complexity of video games, and the harnessing of so many different skills, it is no surprise that making them presents an entirely vast and unique set of problems to overcome throughout the development process in order to attempt to create a 'successful' game. Problems that can pop up during development include, but are not limited to: ensuring fun gameplay, originality, game balancing, monetisation, intuitive controls, choosing a successful genre and platform, gripping storytelling, bug-free, and replayability.

Some of these issues will not apply to this project generally - i.e. marketing and monetisation - however some will also not apply purely due to the nature of the genre - i.e. storytelling.

2.1.1 Genre

One of the first aspects to consider when designing a game would be that of genre. Will it be a roleplaying game? A racing game? A shooter game? A puzzle game? And so on. Genre will define and affect all the rest of the game content, thus it is one of the first hurdles to get through.

Some genres prove to be more popular than others, such as shooters, adventure games, role playing games (RPGs), and platforms steadily holding popularity over time. Series such as Call of Duty (shooter), The Elder Scrolls (RPG), and Mario games (platformer) prove this popularity, with subsequent games in these series holding up to the same success, if not more, as their predecessors. For example, Skyrim is the 13th most sold video game ever (Wikipedia, 2019), despite being the fifth game in the Elder Scrolls series, and there not being a lack of RPGs to choose from. An aspect adding to this popularity however is no doubt the credibility of the games studio; Bethesda (creator of the Elder Scrolls) being one of the most well-known studios to date.

In recent years, battle royales have significantly risen in popularity, with games like Fortnite and PUBG taking the lead. Jumping on the popular genre bandwagon doesn't guarantee success; too many of the same type of game can oversaturate the market, as well as the possibility of the game launching after the hype of such a popular genre has died down.

However, deciding on a niche genre won't necessarily automatically make the game less successful, if it is looked at not purely from a commercial perspective, but instead, say, purely the satisfaction of the player base. More niche genres, or just typically less-popular genres, are fundamentally aimed at a smaller audience, and thus less commercial success is to be expected.

In relation to my project, infinite runners are definitely a more popular genre - typically mobile-based, as touched upon before, or a section as part of a larger wider genre game. Subway Surfers, Temple Run, and Super Mario Run are all well-known games that are part of the infinite runner genre, and all extremely commercially successful; each being within the top 15 most played mobile games ever. Subway Surfers comes in 13th place with 231 million players as of December 2018, Super Mario Run coming in 10th place with 300 million players as of August 2018, and Temple Run coming in 6th place with 500 million players as of September 2013 (Wikipedia, 2019).

Super Mario Run is the only game where it could be argued that its popularity in part is due to the credibility of the company - having being developed by Nintendo; one of the most established games studios in the world - with Temple Run and Subway Surfers instead being their developers' best-known games (Wikipedia, 2019).

2.1.2 Platform

Along with genre, the platform in which the game will be made for is also one of the first aspects to be considered - necessary to decide the development environment, controls, language, and software to be used! The main platforms to choose from are mobile, PC, console, and Virtual Reality (VR) - each coming with their own pros, cons, and restrictions.

Mobile games are typically more casual, less complex, and aimed at a wider audience. They generally allow for lower graphical capacity of the options, and thus are not likely to be of genres such as RPGs and adventure games (which generally boast higher graphics and complex gameplay). As a result of this, mobile games usually are significantly shorter, and thus rely more on replayability, simplicity, and addictiveness to stand out within the crowd of the flooded mobile gaming market. Mentioned in the previous section, Temple Run is a good example of this - the 6th most played mobile game of all time (Wikipedia, 2019), relying on only a basic game structure and graphics, whilst providing a simple, casual, yet addictive gaming experience.

PC games, on the other hand, generally cater towards the more hardcore market of gamers; being home to most, if not all, triple A games. Similar to mobile games, game development for PC has been made so easy and widespread that the gaming market is flooded with multitudes of PC games - varying from low-quality to high-quality. As a result, many well-loved or high-quality games don't receive as much commercial success due to the flooding of the market.

Console games usually come from the same pool as PC games - some successful games are console exclusive; such as Wii Sports being the 4th best-selling game of all time with around 83 million sales (Wikipedia, 2019), however the vast majority are available multi-platform. Unlike PC and mobile game development, console game development is observably less widespread, and thus potentially easier to get your game to stand out. Publishing your game on console can also open it up to a wider set of players; an example being Stardew Valley and Overcooked having released on the Nintendo Switch. With the Switch being a portable console in addition to a home console, this has no doubt attracted more casual players to these games, rather than just the more hardcore PC gamers.

A unique problem to console development however is that of the necessity of backwards compatibility; that is, will the game be able to work on future versions of the console? This is not something decided by the game developers, instead the console developers; and thus might be something to consider when deciding on a platform to develop for. A good example of this would be the Wii U; Nintendo's worst selling console in recent years, with only around 13 million sales (compared to around 34 million for the Switch, and over 100 million for the Wii (Wikipedia, 2019)). Nintendo home console games are not backwards compatible, thus many high-quality Wii U games unfortunately ending up being some of Nintendo's lowest selling home console games; even if they have high player satisfaction. An easy workaround is to merely develop the game for multiple platforms.

Lastly, Virtual Reality (VR). VR development, like console development, is more niche; no doubt due to the higher-end equipment (i.e. VR headsets such as the Oculus Rift or HTC Vive) being a requirement. Again, however, this could be a benefit in the sense that it's easier to get your game to stand out. A downside of VR development would be the audience for VR games is significantly smaller than all previously mentioned platforms; due to the cost of the equipment needed for players.

2.1.3 Gameplay

Gameplay is arguably the most defining aspect of a video game that will determine its success - obviously, and for good reason; it makes up all of the actual game!

The overarching theme and goal for gameplay would be the 'fun' factor. Do players enjoy playing the game? Originality, storytelling, difficulty, features, and controls all tie into this factor.

Whilst originality isn't a necessary ingredient for fun - many games thrive off copying from others - it's definitely a bonus to have at least some splash of originality. What does your game have that others don't? Why will players want to play your game over others of the same type? Super Mario Galaxy by Nintendo (seen in Fig. 2.1) is a good example of originality contributing to its success, receiving a score of 97% on Metacritic (Metacritic, 2019), and 9.7 by IGN (IGN, 2007). Whilst a platforming Mario game at heart, it plays around with physics in unique and interesting ways to capture the players interest and boost the fun factor.

FIGURE 2.1: Gameplay of Super Mario Galaxy (Goomba Stomp, 2016)

Even if original, complex features don't always make for good gameplay however - sometimes, it's better to stick to implementing more conformative ways. No Man's Sky is a good example of falling into this trap - the game received widely negative review on release due to the fact that it simply wasn't fun (Game Rant, 2016). The game was considered boring; although it boasted 'infinite' procedurally generated environments to explore, this meant said environments felt repetitive and empty. In a similar vein to originality is storytelling. Of course, this doesn't apply to all genres of games, however in-depth, likable, characters, a gripping and entertaining plot are typically never looked down upon. For example, this aspect is performed particularly well in The Last Of Us by Naughty Dog - with the game receiving a perfect 10/10 review from Eurogamer (Eurogamer, 2014) - with players and critics alike both raving about and empathising with the games' characters and plot.

Controls are an underlying theme that should also be considered when designing a game. Are the controls intuitive? Are they frustrating? Are they too complex? Non-conforming controls may lessen the fun value to the player; WASD or the arrow keys are usually used for player movement, spacebar for jump, and the escape key for pause. Additionally, the controls should be responsive - the player isn't going to have a good time if there is a noticeable few seconds of delay between pressing a key and the corresponding action being performed. An exception to the rule here would have to be the games I Am Bread, Surgeon Simulator and QWOP - these games thrive on frustrating controls (Pocket Gamer, 2015), with the difficult gameplay only adding to their popularity.

Following on from this is the general difficulty of the game. Too hard and the player might quit in frustration, too easy and the player won't find it challenging enough. Balancing is key to this - I will be discussing this in the next section. A good rule of thumb is to give the player the option to choose their difficulty level, thus allowing them to progress and play at their own pace.

A final point to mention would be the replayability value of the game. How long will the players play this game for? Will they come back to it after completing it (if it can be completed)? Implementing features such as collectibles, achievements, and new game+ (ability to restart the game with higher abilities once its been completed) can attempt to boost replayability value; as well as post-game content (content that is available to the player after finishing the main game). Fallout 4, for example, does this well - this RPG game has multiple winding plot paths the player could take (Game Guides, 2019), encouraging them to play the game multiple times to experience each of these paths (e.g. firstly play as a 'good' character, then secondly play as an 'evil' character).

Genres such as (but not limited to) sports (e.g. FIFA), party (e.g. Super Mario Party), simulators (e.g. The Sims), sandbox (e.g. Minecraft), and building games (e.g. City Skylines) usually have an easier time achieving a high replayability value purely due to the inherent nature of the genres - typically non-linear gameplay with infinite possibilities.

Ultimately, there is no 'one size fits all' for making a fun game - each aspect needs to be considered and tailored carefully in order to fit with the games platform and genre expectations, as discussed in the previous sections.

2.1.4 Balancing

Balancing is an integral part of any game. Easily overlooked, as it primarily consists of just tweaking values, balancing can make or break the game.

Specifically, balancing could involve anything from changing the amount of lives in a game, to the drop rates of loot, to the difficulty of enemies. It can take a great deal of repetitive testing to get the balancing of a game just right; with players of various skill playtesting to provide feedback.

Infinite runner games in particular require balancing for the amount of lives the player has, the speed of the player or road, and the amount of obstacles, power-ups, and currency that spawn. Finely tuning these values are a necessity to make the game feel ‘right’.

2.2 Quality

2.2.1 Graphics

Graphics are a vital component to video games; although this does not necessarily mean the graphical detail has to be high. Many popular video games don’t have super realistic or detailed graphics at all, instead the graphical quality preferred to be consistent, aesthetically pleasing, and fitting with the overall style and theme of the game. To name one, Candy Crush (seen in Fig. 2.2), the 3rd most popular mobile game by players, makes use of charming, simplistic, and cartoon-style graphics that fit its overall theme.

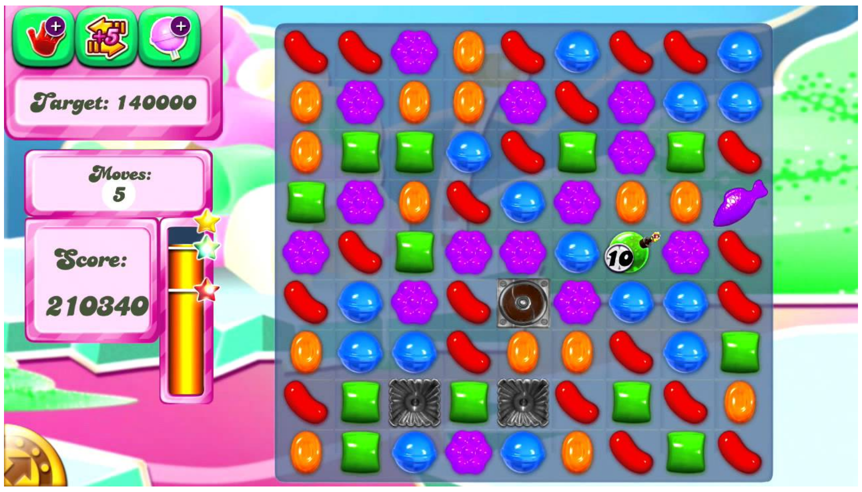


FIGURE 2.2: Candy Crush graphical design (YouTube, 2014)

Generally, it is better to make less-realistic, more stylistic graphics well, than realistic graphics badly - only triple A studios are likely to possess the skill and manpower necessary to succeed in making quality realistic graphics. Some examples of such games would be the Assassin’s Creed and Tomb Raider series by Ubisoft, and Detroit: Become Human by Quantic Dream. The latter of these games was shot using motion capture to achieve such realistic animations (seen in Fig. 2.3); a costly process that would likely not be available to smaller studios.

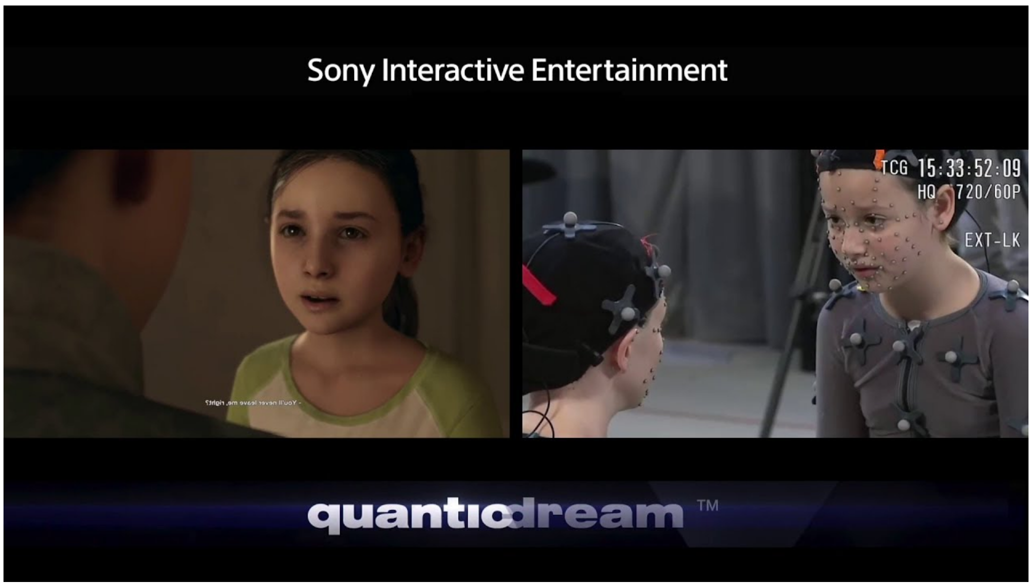


FIGURE 2.3: Motion capture behind-the-scenes and finished graphical comparison for Detroit: Become Human (YouTube, 2018)

2.2.2 Bugs

An obvious aspect to making a successful game would be to minimise bugs! All parts of the game should ideally be working correctly, as intended. Game studios make user of quality assurance and playtesting in an attempt to weed out any possible bugs or unintended behaviour, thus informing the developers to fix and remove such bugs or behaviour.

Both larger and smaller game studios can fall prey to publishing an unsuccessful game due to bugs; namely Assassin's Creed Unity, being infamous for its graphical bugs on release (seen in Fig. 2.4), and more recently, Fallout 76 - rated a mere 52% by Metacritic (Metacritic, 2019), primarily due to its abundance of buggy gameplay.

One way to prevent buggy releases is merely giving the game development more time to find and fix these bugs. Additionally, some games release betas (unfinished pre-release of game content) in order to recruit the wider community into providing feedback and finding bugs that may otherwise have been missed. For example, a 'Mining and Smithing' beta (Jagex, 2018) was released in the Massive Multiplayer Online (MMO) game RuneScape; as it covered a significant update to one of the core skills in the game, and thus the developers wanted to ensure they overlook as little as possible.

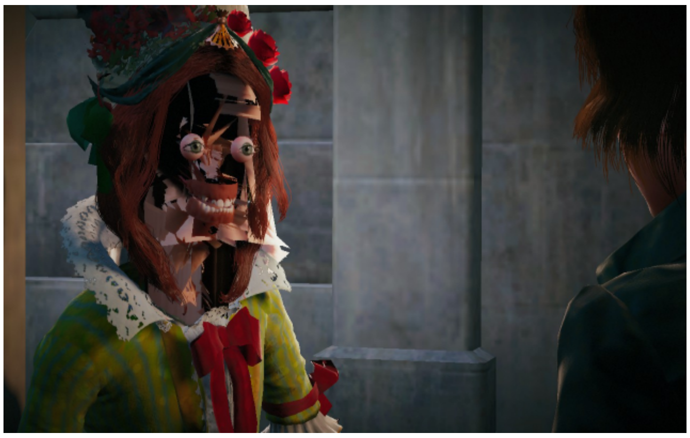


FIGURE 2.4: Graphical bug in Assassin's Creed: Unity (Ars Technica, 2014)

On the flip side, 'buggy' gameplay can also prove to boost a games popularity - Goat Simulator is classic example of this, with its unpredictable ragdoll physics adding to its charm and amusement (seen in Fig. 2.5) However, this is the exception and not the rule - generally games should not aim to have buggy gameplay.

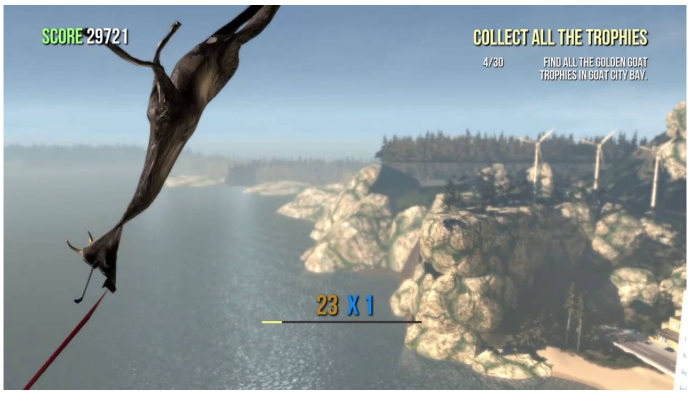


FIGURE 2.5: Ragdoll physics in Goat Simulator (YouTube, 2016)

2.3 Business

2.3.1 Monetisation

Both a business aspect, and a gameplay aspect, is that of monetisation. Monetisation can come in various forms, such as microtransactions (MTX) within the game itself, or having to pay to access the game; in a subscription-based format.

An example of a popular microtransaction monetisation model would be that of Fortnite - the game itself is free, with Epic Games (the developer) primarily making money from selling 'skins'; purely cosmetic outfits and weapons for your character.

Subscription based games on the other hand, can offer either a subscription for access to wider gameplay - such as RuneScape's subscription model - or requiring a subscription to access any gameplay - such as World of Warcraft. The benefit of having at least some of the game available for free is that it entices players to buy a subscription who may have not otherwise done so, had they not been able to 'demo' the game to see if they enjoyed it.

Monetisation is generally necessary in some format for larger studios with ad-free games, as the developers need to be paid for their work. The problem faced here is balancing the pricing and content of the monetisation; too lowly priced and it's not reasonably for the amount of development work put in for creation, too highly priced and few people will purchase it - as per Star Wars Battlefront II, which faced massive backlash over excessively priced microtransactions back in 2017 (GameSpot, 2017).

Additionally, in terms of microtransaction monetisation, the content being sold should, as a rule, not boost gameplay - this has shown time and time again to be unpopular with players due to its unfair advantage - or be too rare to win, if the microtransaction in question is a loot box.

Microtransaction advertising should not be too 'in your face' either - players may end up ignoring it entirely, or be put off from playing the game at all if they feel pressured into spending extra money.

2.3.2 Marketing

Marketing is yet another key aspect of releasing a successful game - although arguably not a direct problem of game development itself, it is worth a mention due to it coming hand-in-hand.

Whether it be paid promotions, commercials, or word-of-mouth, marketing is the way that games mostly get discovered. Like it or not, good marketing is generally necessary for a successful game; good marketing can consist of appealing adverts and targeting the right people who will want to play this game.

A unfortunate point to mention in regards to marketing is that is it much easier for larger companies. Not only can these companies afford more commercial material, they will already likely have credibility with many players from previous games; thus having a lesser need for advertising in comparison to smaller development studios. Many indie studios and developers combat this through utilising social media; it's cheaper to pay YouTube and Twitch influencers in an attempt to spread word-of mouth for your game, than it is to purchase bulletin boards all across a city, or adverts on the TV. For example, Five Nights At Freddy's (a survival horror game), developed by one man, Scott Cawthon, gained a cult following after popular Youtuber PewDiePie played the game on his channel back in 2014 (P Hernandez, 2015).

2.4 Post-Release

Unsurprisingly, a not-often-considered aspect when developing a game is that of post-release content. As this comes after the game is released, and thus after it's usually been deciphered whether the game was a success or not, post-release content is typically not one of the first problematic aspects that comes to mind when I think of game development - specifically for non-subscription games.

Subscription games do face this problem, however in a different way. Rather than being expected to release a few post-release downloadable content packs, these games players rely on them being consistently updated on at least a semi-regular basis. Likewise, certain online games (not necessarily subscription-based - for example, Fallout 76) also hold this expectation.

Post-release content, be it free or paid for, a time-limited event or permanent, can bring back inactive players to the game; potentially boosting sales despite not having a recent release. A vast majority of popular games thus utilise post-release downloadable content; Grand Theft Auto V, despite having released back in 2013, was the 11th best selling game in 2018 (Business Insider, 2019) - no doubt due to the constant free updates its online mode receives, keeping the player base active and thus the game successful.

2.5 Conclusion

To conclude this chapter, there are endless possible problems to consider when developing a game. Whilst the perceivable best decision at the time can be made for the development of the game, it does not necessarily equate to success in either commercial or player satisfaction aspects.

Being a well-known games studio, minimising bugs, getting the balancing just right, providing fun gameplay with high replayability value, and having consistent thematic graphics, are all strong characteristics that increase the chance of releasing a prosperous video game.

Chapter 3

Specifications

In a general sense, I set out to create a fully-working, third person, 3D, infinite runner-style game for PC, using the Unity game engine, set around dog walking. The game's assets will be made mostly, if not all, by myself, using Blender for modelling and animation - with the art style following that of colourful and cartoon-ish.

All of the following requirement sections additionally come under the presumptive requirements of the system having all parts working with no errors, for PC.

3.1 Gameplay Requirements

TABLE 3.1: Gameplay requirements

#	Requirement
1	Enjoyable to play overall
2	Ability to explore hub scene, with appropriate collisions and interactions
3	Ability to access infinite runner levels from various points in the hub scene
4	Infinite runner levels with moving roads, randomly spawned obstacles or power-ups, life system, and score system
5	Appropriate balancing - in terms of level difficulty, overall player control sensitivity/speed, and AI action weighting
6	Reward system - hat shop, ability to purchase hats for stars

1. This requirement is the most basic underlying demand in order for this project to succeed. Games, at their core, are designed to be fun, with this having to be considered in relation to all of the other requirements.

2. The hub scene will be the main area of the game, and thus is a basic requirement that needs to be implemented for the game to succeed. The vast majority of the game is based around and within the hub itself, so this is a necessary aim to achieve, otherwise the project shall be considered a failure. Appropriate collisions and interactions should be implemented, to meet the 'all parts working' requirement - the player should not be able to access out of bounds areas, or clip through any models.

3. In order to be able to play the infinite runner levels, they must be accessed from the hub scene. If this requirement is not met, the levels will thus not be playable; a core aspect of the project aims.

4. Following on from the previous requirement, the levels themselves must be playable. The levels must have a moving floor, with obstacles and power-ups in order to fit with the infinite runner genre. Additionally, there must be a form of life and scoring system in order to determine level progress and success.

5. The game must have appropriate balancing in order for it to be accessible and appealing to players. The control sensitivity should feel normal - not too fast, or too slow (potentially frustrating).

   There should be varying level difficulties to provide players the option of what pace they want to play the game at; these difficulties should have corresponding appropriate values, with easy being accessible to non-gamers, and hard being a challenge for experienced gamers. The values affected may be the road speed, player speed, rarity of power-ups and obstacles, spawn rate of power-ups and obstacles, and the scores required in order to earn star ratings.

   Furthermore, the AI should have their actions appropriately weighted in order to present a more natural front.

6. In order to boost replayability value for the game, there should be a form of reward system in order to encourage the player to replay the various levels. This form will be that of a hat shop, where the player can thus purchase cosmetic hats in trade for stars they have earned within levels.

3.2 AI Requirements

TABLE 3.2: AI requirements

#	Requirement
1	Ability to perform a variety of actions: idling, walking, running, and flocking
2	Ability to perform actions in a weighted fashion
3	Ability to gain a dog that follows the player upon interacting with the dog AI
4	Ability to express discontent upon collision with the player

1. The AI must be able to carry out various actions; at least one, else it would not be an AI. Ideally it should be able to perform multiple actions, so it more naturally fits into its environment, and puts it a step above the most basic possible level of AI. Idling, walking, running, and flocking, are all natural behaviours for an AI that is based in a suburban town setting. Idling should involve the AI doing nothing for a set period of time; standing still (in idle animation). Walking and running should involve guiding the AI to a certain destination on the map - with it either walking or running to said destination. Flocking should involve guiding the AI to another AI within the scene.

2. These actions should be performed in a weighted manner. Each action should have a percentage chance of being chosen. More commonplace and thus natural actions, namely idling and walking, should have a higher percentage of happening than those of running and flocking.

3. Interacting with a dog AI should provide the player with a dog companion of their own. This companion will have no abilities of its own, and purely serve as a cosmetic feature.

4. Upon the player colliding with a human AI, the AI should express discontent in form of a text box appearing above them, with a sound effect. Initially, the AI may merely say a 'Hello', however after increasingly higher collision amounts, the AI should express their discontent with the player bumping into them - for example, saying 'Leave me alone!'.

3.3 UI Requirements

TABLE 3.3: UI requirements

#	Requirement
1	Overall consistent, thematic, and stylistic design
2	Start menu - capable of starting or quitting the game
3	Pause menu - capable of pausing game from hub or levels, and quitting the game
4	Bulletin menu - contains details about controls and gameplay to inform the player
5	Level UI - capable of displaying updated lives and score

1. All of the user interfaces should follow a consistent theme, with a colourful and cartoon style. The designs should overall be intuitive (obvious labelling etc.) and easy to use, as well as aesthetically pleasing.

2. The start menu should be a friendly welcome into the game, as it will be the first thing the player sees and thus sets the scene, providing a straightforward way to start the game, or quit the game.

3. The pause menu should be a simple interface, purely to serve as a way to pause the game, and quit the game. This should be available to access from both the hub, and the level scenes, so the player is unrestricted in where they can pause and quit the game from (provide them the freedom to play at their pace).

4. The bulletin menu will be the interface that displays important information about the game, including instructions, about, and controls. This should provide all the necessary details the player requires, in a concise and readable format, in order to understand how to play the game.

5. Within the levels, there should be a level UI that displays the current score and number of lives the player has. This should be constantly updated in real

time, to keep the player informed of how they are currently doing. It should be simple and out-of-the-way, as to not cover up gameplay, yet still easily visible at a quick glance - thus with appropriate contrasting colouring and font size.

3.4 Graphical Requirements

TABLE 3.4: Graphical requirements

#	Requirement
1	Overall consistent, thematic, and stylistic design
2	Appropriate sizing and placement of models
3	Believable walk, run, and idle animation cycles

1. Identical to the UI requirements, and thus matching the interface designs, the graphics of the game should also follow a consistent theme - an aesthetically pleasing cartoon-ish and colourful style.
2. All of the models should be placed appropriately around the game world. The models should all be sized to a consistent scale, and set down to decorate the environments in a natural fashion. This requirement will create an environment that will primarily help shape what the player should feel and experience when playing the game; setting the scene.
3. Lastly, the character models should all possess believable walk, run, and idle animation cycles to contribute to giving the game a more polished and finished feel. Any other possible animations for models - such as foliage - should also meet this requirement.

Although not graphics, audio should also follow a consistent and thematic design, and placed around appropriately.

Chapter 4

Design and Implementation

In this chapter, I will be discussing how I initially designed each specific feature for the game, and how they were ultimately implemented.

Firstly, I decided to choose Unity as my game engine of choice for various reason. Unity is one of, if not the, most well-known, free, game engines available alongside Unreal); and for good reason. It receives constant updates, has a huge array of features, and there is a mass of learning material available online to help with learning the software.

Secondly, I went with Blender as my software of choice for modelling and animation, again due to being one of the most widely-known, free, computer graphics software available (alongside Maya). Although it arguably has a steep learning curve, again, there are copious resources available online to aid with this process.

To preface this section is a description of the game to help your understanding. The game primarily takes place in a suburban town, with 3 areas: town, park, suburbs. The player spawns in the town area upon starting the game from the start menu, where they then gain free roam to do as they wish. Various level indicators - which transport the player to infinite runner-style levels - can be found around this 'hub' scene, as well as wandering dog and human AIs. Within the levels, the player (starting with 3 lives) must avoid obstacles (collision with any results in a life lost) and gain power-ups (to a maximum of 5 lives). Upon losing all their lives, the player is faced with a level over interface, with the option to take them back to the hub. The game can be paused at any time, to quit the application.

A flowchart of the scene managing process can be seen below in Figure 4.1.

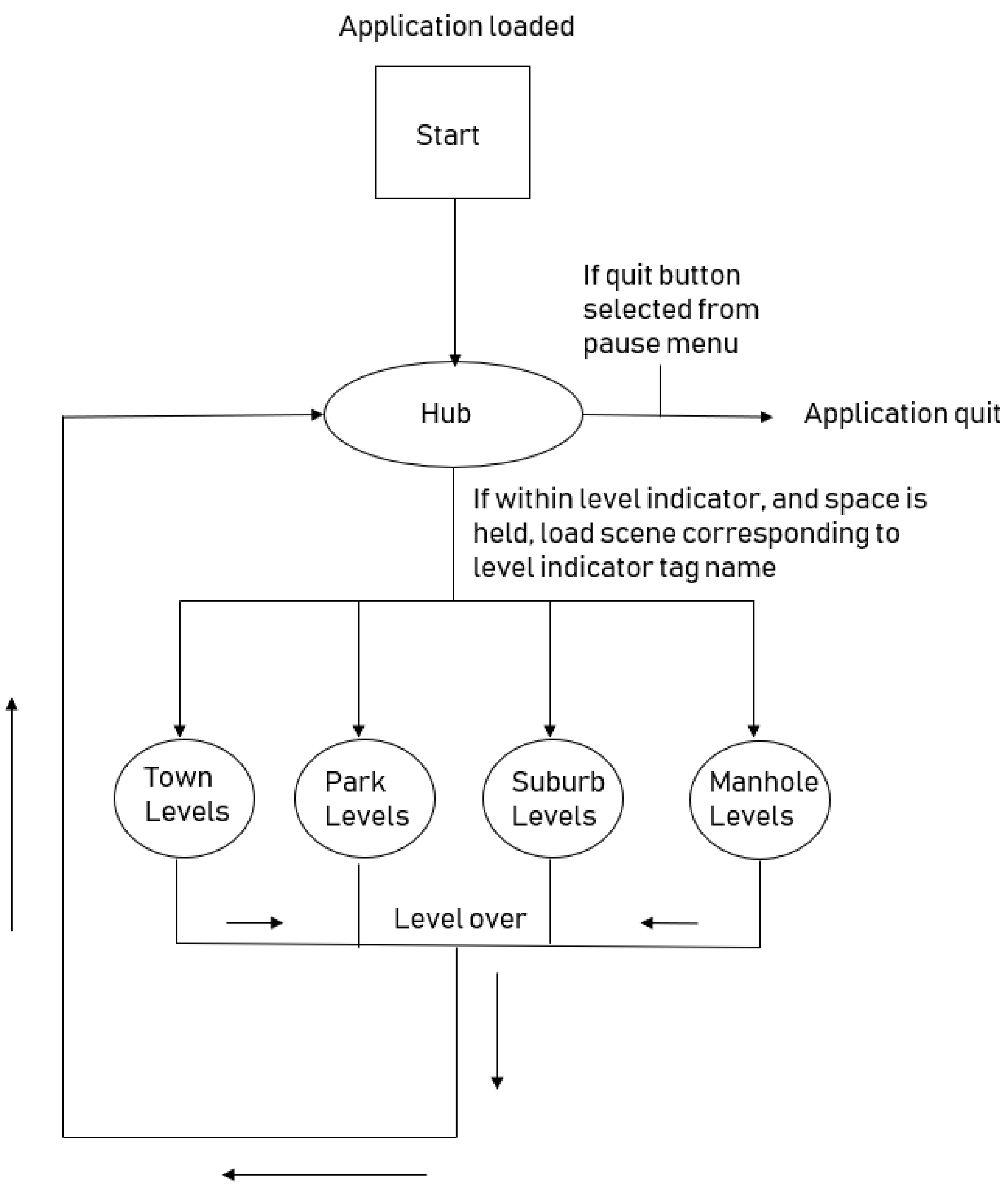

Application loaded
Start
If quit button selected from pause menu
Hub
Application quit
If within level indicator, and space is held, load scene corresponding to level indicator tag name
Town Levels
Park Levels
Suburb Levels
Manhole Levels
Level over

FIGURE 4.1: A flowchart displaying the scene flow of the game

4.1 Player Controls

Relevant scripts for this section: PlayerController.cs, LevelPlayerController.cs, LevelLoad.cs, AIDogBehaviour.cs, PauseMenu.cs , LevelPauseMenu.cs, BulletinUI.cs

The player is capable of performing various actions at the press of a key, including:

- Forward movement
- Backwards movement
- Rotate camera left

- Rotate camera right
- Pause game
- View mode
- Interactions

The most familiar controls for player movement is that of WASD and the arrow keys, and thus why I chose them for the player movement. Initially, I only had implemented the arrow keys - I considered this might be more intuitive to a wider, non-gaming, crowd - however, player feedback led to the additional implementation of WASD. This provides players with the choice to use their key of preference.

Contrary to various other games, there is no general mouse control for the player - only within view mode. I opted to exclude this due to preferring keeping the controls simple, and thus arguably easier to get used to. As the gameplay and features are not extremely complex, the project doesn't call for a need for more controls.

Pause mode can be activated with either the P key, or the escape key. Again, initially I had only implemented the P key - P for pause - however ended up implementing the Escape key additionally due to player feedback as it's akin to WASD in terms of commonplace use within video games.

View mode is a feature that allows the player to freely look around the game world. I decided to add this in at a later stage of development to provide the player with a less restrictive way of viewing their surroundings; after finding myself wanting to look at disallowed angles whilst playtesting. The V key was chosen to toggle this on or off due to 'view' beginning with V; the player is less likely to forget the key if it corresponds to the name of the feature (e.g. I for inventory).

The player can interact with various features throughout the game hub, including the dog AI (to gain a dog companion), loading the bulletin board interface, and loading the levels. Interaction is carried out by holding down the spacebar - another intuitive control within video games. The key must be held down for a short time as to not potentially irritate the player should they suddenly change their mind (and thus not want to carry out the interaction), or if they accidentally press it; this ensures the player actually wants to carry out the interaction.

4.1.1 Player Movement

Player movement is implemented in the update function (checks condition every frame) in PlayerController.cs and LevelPlayerController.cs with the following steps:

1. Checking whether the appropriate key is held down - W/up for forward, S/down for backwards, A/left for camera rotate left (hub only), D/right for camera rotate right (hub only)
2. If appropriate key is down, checking that pause mode is false, and view mode is false (hub only)
3. If conditions met, setting corresponding animator parameters to appropriate value (run for forwards, walk for backwards - hub only), transforming the player location in the corresponding direction * real time * speed (integer variable), and activating the footstep audio (hub only, footsteps constantly activated within levels)

For example, the code for forward movement can be seen below, in figure 4.2:

```
if ((Input.GetKey(KeyCode.UpArrow) || Input.GetKey(KeyCode.W)
    ) && viewMode == false && PauseMenu.pause == false)
        {
            move.SetInteger("RunOn", 1);
            footstep.SetActive(true);
            transform.Translate(Vector3.forward * runSpeed *
                Time.deltaTime);

            if (hasDog == true)
            {
                dogMove.SetInteger("RunOn", 1);
            }
        }
```

FIGURE 4.2: The code used to move the player forward

The animator workflow for both the player, and the dog (if the player has one), can be seen below, in figure 4.

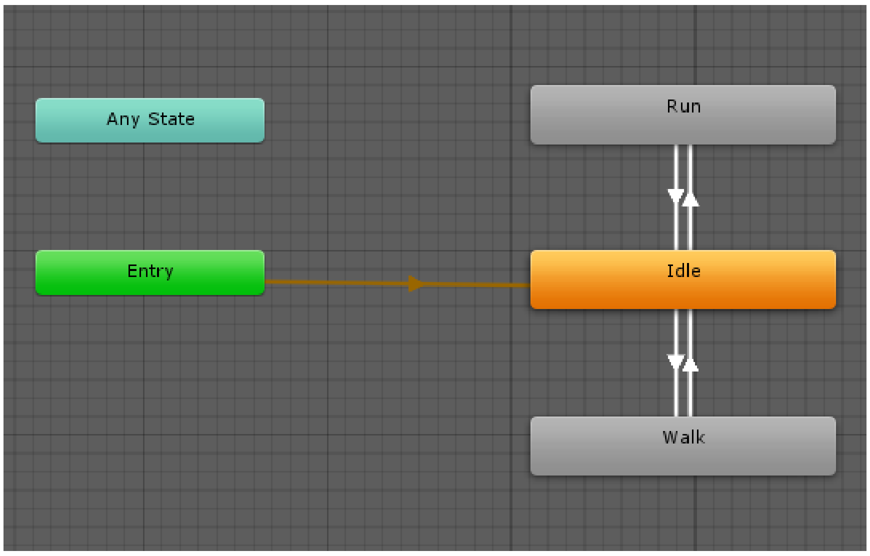


FIGURE 4.3: A view of the player animator controller

Additionally, should the player reach an out-of-bounds area (i.e. they breach the colliders), the game will teleport the player back to the spawn area. This is carried out by the game constantly checking if the players' position coordinates are within the allowed coordinates, as seen in figure 4 below. Although there are colliders surrounding the playable area, this is a safety net feature implemented as a result of playtesters succeeding in clipping through certain colliders.

4.1.2 Other Controls

Boolean variables exist for both view and pause mode to keep track of the current state of the game; both are initialised with a false value, which only changes to true upon carrying out the appropriate action which toggle the modes on (as detailed below).

Pause

Pause mode provides a way to pause the game from any point. It works by setting the Pause UI active (game is loaded with it inactive), and the time scale of the game to 0 (thus freezing time).

```
if ((Input.GetKeyDown(KeyCode.P) || Input.GetKeyDown(KeyCode.
    Escape)) && BulletinUI.isBulletinOn == false && viewMode.
    viewMode == false)
        {
            if (pause == false)
            {
                pause = true;
                pauseUI.SetActive(true);
                Time.timeScale = 0.0f;
            }
            else
            {
                resumeButton();
            }
        }
```

FIGURE 4.4: Code showing what happens when the game is paused

The mode is toggled on within PauseMenu.cs, and LevelPauseMenu.cs (for both the hub and level scenes, respectively) if the player presses down the 'P' or 'Escape' key, and the following conditions are met: pause mode isn't currently true, view mode isn't currently true (PauseMenu.cs only), and the bulletin interface isn't currently being accessed (PauseMenu.cs only). If the game is currently paused, the mode can be toggled off by pressing the same keys.

View Mode

View mode, only available in the hub scene, utilises a script found I online - this isn't a key feature, and I found this script to work extremely well, and thus did not attempt to implement it myself (Learn Everything Fast, 2017). This mode enables the player to look around at any angle using the mouse, at the cost of disabling player movement while it's true.

The mode is toggled on within PlayerController.cs if the player presses down the 'V' key, and the following conditions are met: view mode isn't currently true, pause mode isn't currently true. If the game is currently in view mode, the mode can be toggled off by pressing the same keys (resetting the camera to as it were).

```
if (Input.GetKeyDown(KeyCode.V))
        {
            if (viewMode == false && PauseMenu.pause == false
               )
            {
                viewMode = true;
                cameraMouse.GetComponent<MouseCamera>().
                   enabled = true;
            }
            else
            {
                viewMode = false;
                cameraMouse.GetComponent<MouseCamera>().
                   enabled = false;
                camY = this.transform.eulerAngles.y;
                cameraMouse.transform.eulerAngles = new
                   Vector3(camX, camY, camZ);
            }
        }
```

FIGURE 4.5: View Mode toggle code

4.1.3 Player Interaction

The interaction control is used for both loading levels - once the player is within a level indicator - and gaining a dog companion.

Level Loading

Player interaction with loading levels and bulletin interface is implemented in LevelLoad.cs and BulletinUI.cs with the following steps:

1. Check if the player is within distance of the level indicator (if radius of level indicator <= distance)
2. If within distance, check if space is pressed
3. If space is pressed, start counting time - *or load bulletin interface*
4. If counter breaches a couple of seconds, and space is still held down (timer variable resets if space is let go of), turn on timer, set fade in animation and start music active
5. Once timer has reached a few seconds, load corresponding scene

The timer aspect of this, and other parts of the project, is necessary in order to give the fade in and fade out animations time to fully execute (else the next scene is loaded at the same time the animation is set to start). I added animations to make the transition between scenes look more polished.

```
if (distance <= particleRadius)
        {
            if (Input.GetKey(KeyCode.Space))
            {
                keyHoldTime += Time.deltaTime;
            }

            if (Input.GetKey(KeyCode.Space) && keyHoldTime >=
                1f)
            {
                timerOn = true;
                fadeIn.SetActive(true);
                startMusic.SetActive(true);
            }
        }
```

FIGURE 4.6: Trigger code for level load

This aspect will be followed up on in section 4.3.1 (Levels: Level Loading).

Dog Companion

The player can gain a dog companion through pressing space whilst colliding with a dog AI. This companion is a child game object of the player, that is set from inactive to active upon meeting the following conditions:
In AIDogBehaviour.cs:

1. Check if a dog AI is currently colliding with the player
2. If colliding, check if space is held down, and the player doesn't already have a dog (initialised false variable hasDog)
3. If both true, change PlayerController dogType variable to correspond with the name of the dog AI (dogType initialised with no value), and set hasDog to true

```
void OnCollisionStay(Collision collider)
    {
        if (collider.gameObject.name == "Player")
        {
            if (Input.GetKeyDown(KeyCode.Space) &&
               PlayerController.hasDog == false)
            {
                PlayerController.hasDog = true;
                switch (this.name)
                {
                    case "Dog(Clone)":
                        PlayerController.dogType = 1;
                        break;
                    case "Dog (black)(Clone)":
                        PlayerController.dogType = 2;
                        break;
                    case "Dog (blonde)(Clone)":
                        PlayerController.dogType = 3;
                        break;
                    case "Dog (grey)(Clone)":
                        PlayerController.dogType = 4;
                        break;
                    case "Dog (white)(Clone)":
                        PlayerController.dogType = 5;
                        break;
                }
            }
        }
    }
```

FIGURE 4.7: Code for gaining a dog companion - in dog behaviour

Then, in PlayerController.cs:

1. Check if hasDog is true, and dogCount (initialised int at 0) is 0 (meaning the player doesn't already have a dog)
2. If true, add 1 to dogCount, set a woof sound effect active, set the corresponding dog child object active, and set the dog animator to the dog child

```
if (hasDog == true && dogCount == 0)
            {
                woof.SetActive(true);
                dogCount += 1;
                switch (dogType)
                {
                    case 1:
                        dog.SetActive(true);
                        dogMove = dog.GetComponentInChildren<
                            Animator>();
                        break;
                    case 2:
                        dogBlack.SetActive(true);
                        dogMove = dogBlack.
                            GetComponentInChildren<Animator>()
                            ;
                        break;
                    case 3:
                        dogBlonde.SetActive(true);
                        dogMove = dogBlonde.
                            GetComponentInChildren<Animator>()
                            ;
                        break;
                    case 4:
                        dogGrey.SetActive(true);
                        dogMove = dogGrey.
                            GetComponentInChildren<Animator>()
                            ;
                        break;
                    case 5:
                        dogWhite.SetActive(true);
                        dogMove = dogWhite.
                            GetComponentInChildren<Animator>()
                            ;
                        break;
                }
            }
```

FIGURE 4.8: Code for gaining a dog companion - in player controller

Gaining a dog companion was a feature added as result of playtesting; multiple players queried whether they could have a dog in the hub scene, as they had one within the level scene. The game is based around dog walking, thus it makes sense to provide the player with the ability to have their own dog. A variety of dog colours are available, to not only add variety and aesthetic value, but give the player the choice to choose their colour of preference.

4.2 AI

Relevant scripts for this section: AIBehaviour.cs, AIDogBehaviour.cs, AISpawn.cs

The AI in this project uses a finite-state machine-like implementation. I opted for finite state machines, over more complex methods such as behaviour trees, due to a combination of time constraints (AI is not the core necessity of the project), and the needs for the game - AIs walking around town don't particularly require very advanced actions.

This section refers to both the human and the dog AIs.

4.2.1 Spawning

AISpawn.cs (applied to the terrain of the environment) determines the spawn coordinates for each AI, number of AI to spawn, and what each model for the AI will be. This is done with the following steps:

1. Placing all the possible models for both dogs and humans into arrays
2. Generating a random value between 15-25, to determine how many AI to spawn
3. For loop through this value, doing the following each loop:
4. Generate random X, Z, and Q (q for quaternion - value for random Y rotation) values
5. Generate random value between 0-6 (or 0-5 for dogs) to determine what model will be used
6. Instantiate human/dog with the value in step 5 as model array index, and values in step 4 as vector coordinates and euler coordinate

```
for (int i = 0; i < spawnCount; i++)
        {
            spawnX = Random.Range(-28f, -10f);
            spawnZ = Random.Range(-6f, 72f);
            spawnQ = Random.Range(0f, 360f);
            humanI = Random.Range(0, 6);
            Instantiate(humans[humanI], new Vector3(spawnX,
                0.3f, spawnZ), Quaternion.Euler(0f, spawnQ, 0f
                ));
        }
```

FIGURE 4.9: Code for spawning AI - humans only

Spawning is implemented in this way to provide variation in gameplay - rather than statically spawning the same models in the same places on each load.

4.2.2 Behaviour

A flowchart displaying an overview of the actions of the AI can be seen below in Figure 4:

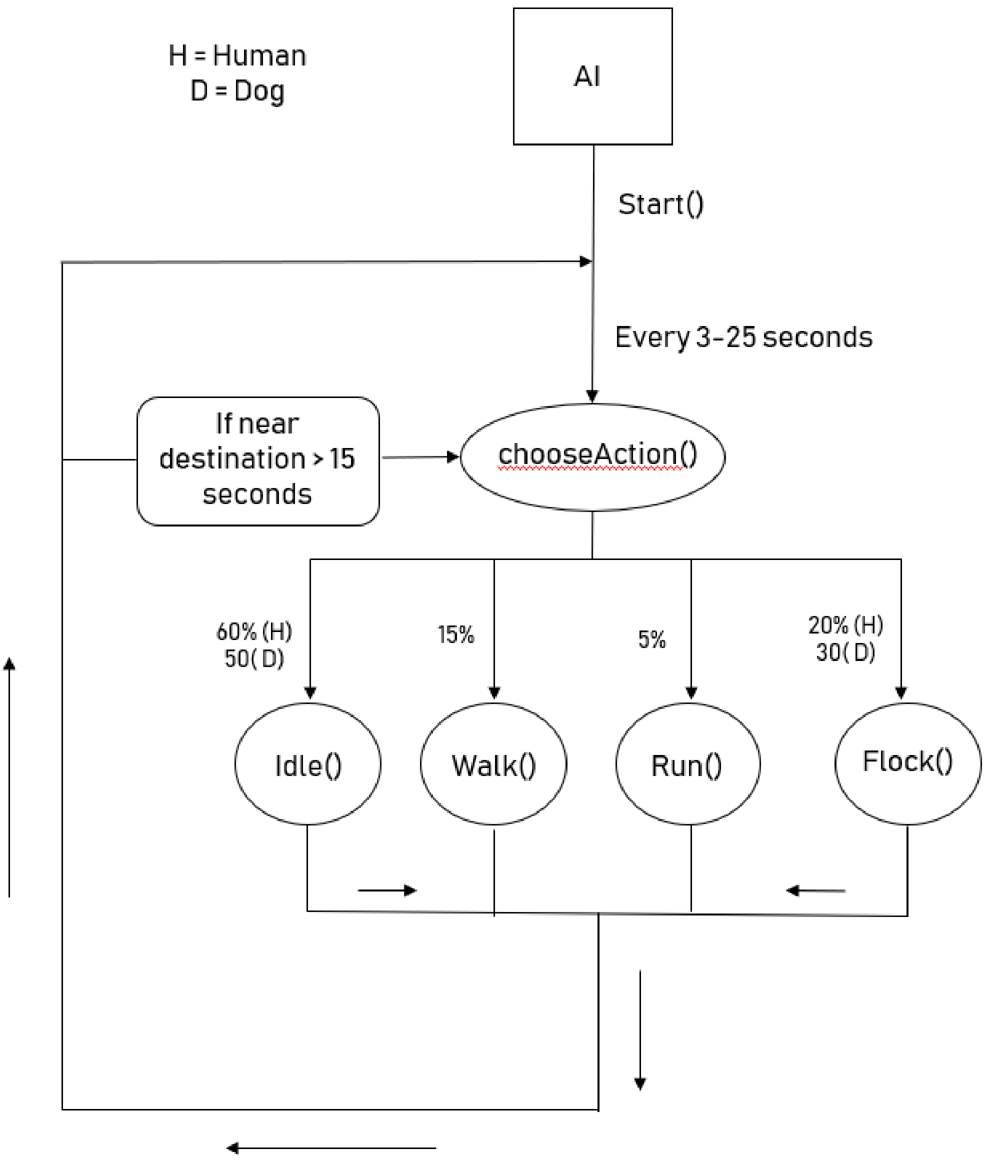


FIGURE 4.10: A flowchart showing the actions and weighting the AI can take

Choose Action

As seen in the flowchart, the chooseAction() function is rolled every 3-25 seconds if the AI is stationary (preventing them from abandoning their current destination) to guide the AI as to their next activity.

The weighting behind each action roll is justified through the following means:

- Idle should be a high chance, to provide a more natural environment, rather than all of the AI moving around constantly
- Walk should be a higher chance than run, reflecting natural behaviour in real life (people tend to not run to their destinations in public!)
- Flock should still have a fairly reasonable chance, to actually be noticeable - rather than indifferentiable from walking or running (also, higher chance for dogs due to wild dogs more likely to hang out in a pack than humans)

Idling, Walking, Running

If idling is selected, simply nothing happens - a free roll.

If walking or running are selected, random X and Z coordinates within a specified area (the accessible game map) are rolled to determine a vector3 destination for the AI. This vector is passed to the SetDestination function; utilising Unity's NavMesh feature (Github, 2019) - an add-on package for AI that provides simple pathfinding and utilities. This can be seen in action below, in figure 4.11:

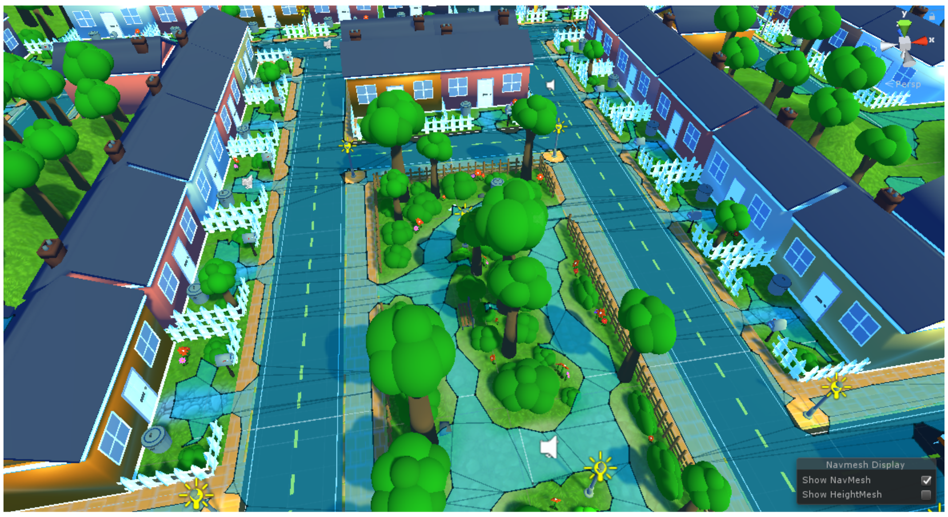
```
void walk()
    {
        nav.speed = 1;
        Vector3 walkDestination;
        float destX = Random.Range(-28f, -10f);
        float destZ = Random.Range(-6f, 72f);
        walkDestination = new Vector3(destX, 0.3f, destZ);
        destination = walkDestination;
        nav.SetDestination(walkDestination);
    }
```

FIGURE 4.11: Code for AI walk & run function (run is mostly identical)

If the action is running, the AIs speed is increased, and triggers their run animation on. Walk action, on the other hand, sets the AI to a lower speed, triggering their walk animation on. The velocity of the AIs NavMesh is constantly checked to determine which animation should currently be active.

I opted to use Unity's NavMesh feature for pathfinding again due to time constraints - if there's a perfectly well usable function readily available to use, I'm not going to spend time reinventing the wheel.

A point to note, is that before the NavMesh system can be used, the NavMesh-Surface function had to be implemented in the environment; to inform the AI where and where they could not access, as seen below, in figure 4.12:



Navmesh Display
Show NavMesh ✓
Show HeightMesh ☐

FIGURE 4.12: A view of the navmesh - displaying AI walkable areas in blue

Flocking

Flocking, named after flocking behaviour, was chosen as an action for the AI primarily due to humans being social creatures, and dogs being pack animals. This also provided an opportunity to implement a more advanced action, in comparison to mere walking.

This function works by finding the nearest AI, and setting their position as the current AIs destination, as follows:

1. Create array of all other humans/dogs within the game ("FindWithTag")
2. Create a new array of the same length, and loop through the array in step 1, filling each value with that of the distance between each human/dog and the current AI
3. Duplicate the array from step 2, and use a bubble sort algorithm to determine the distances from lowest to highest
4. If the lowest value is within 5f (i.e. too close - flocking wouldn't make a notable difference), increase the lowest value to the next lowest value in the array (if there are no AIs that aren't too close the function is broken out of)
5. Set the AIs destination to that of the AI position that corresponds to the lowest value (as determined from steps 3 and 4).

Step 5 is calculated using the following code, with *distances* being the array in step 2, and *humans* being the array in step 1 (nav = reference to navmesh component):

```
for (int i = 0; i < distances.Length; i++)
        {
            if (lowest == distances[i])
            {
                target = humans[i];
                flockDestination = humans[i].transform.
                    position;
            }
        }

        destination = flockDestination;
        nav.SetDestination(flockDestination);
```

FIGURE 4.13: Part of the code for the flock() AI action

Destination Reset

As a precautionary measure, to prevent AIs from infinitely attempting to reach their destination even if not possible, a reset method is also implemented within the AI scripts.

Located in the update function, this feature constantly checks this distance from the AIs current destination to the AI itself. A timer is started once they're within a short distance (3f), and the path is reset with a new action rolled if they've been within this short distance of their destination for around 10 seconds - shown below in figure 4 (nav = navmesh component):

```
if (Vector3.Distance(destination, this.transform.position) <
    3f)
        {
            timer1 += Time.deltaTime;
            if (timer1 > 15f)
            {
                nav.ResetPath();
                chooseAction();
                timer1 = 0f;
            }
        }
```

FIGURE 4.14: Code that resets the AIs current destination

4.3 Levels

Relevant scripts for this section: LevelLoad.cs, MoveFloor.cs, SpawnBehaviour.cs

4.3.1 Level Loading

Touched upon in section 4.1.3, in addition to simply triggering the level scene, the levels also require knowledge of the difficulty value and which scene setting to load - be it the town level scenes, park level scenes, or suburban level scenes.

Scene setting is determined through checking the tag of the level indicator in LevelLoad.cs - each level indicator object has a tag corresponding to its location (i.e. indicator within town are tagged "Town", park tagged "Park", and suburb tagged "Suburb"). A switch statement, as shown below, implements this:

```
switch (this.tag)
                {
                    case "suburb":
                        levelSetting = "SuburbLevels";
                        break;
                    case "park":
                        levelSetting = "ParkLevels";
                        break;
                    case "town":
                        levelSetting = "TownLevels";
                        break;
                    case "manhole":
                        levelSetting = "ManholeLevels";
                        break;
                }
```

FIGURE 4.15: The switch statement that determines which level setting to load

Difficulty, however, is less straightword in its implementation. Within Player-Controller.cs, all of the level indicator objects are looped through, and if the player is within distance of any of them, the corresponding difficulty value is saved to that script - in a public static variable, enabling it to be accessed from the level scene. This loop can be seen below in figure 4.16:

```
for (int i = 0; i < levels.Length; i++)
            {
                distance = Vector3.Distance(levels[i].
                    transform.position, this.transform.
                    position);
                if (distance <= particleRadius)
                {
                     if (i >= 0 && i <= 2)
                     {
                         difficulty = 1; //easy difficulty
                             value
                     }
                     else if (i >= 3 && i <= 5)
                     {
                         difficulty = 2; //medium difficulty
                             value
                     }
                     else if (i >= 6 && i <= 8)
                     {
                         difficulty = 3; //hardest difficulty
                             value
                     }
                }
            }
```

FIGURE 4.16: The code that determines the difficulty value of the level to load

Easy levels are stored in the first 3 positions of the levels array, medium stored in the next 3, and hard stored in the final 3.

Once the appropriate level has loaded, MoveFloor.cs and LevelPlayerController.cs retrieve the difficulty value from PlayerController.cs to determine how to alter their variables in regards to difficulty. The following variables are affected by difficulty:

- Player speed (harder = slower)
- Road speed (harder = faster)
- Item spawn (harder = spawn more often)

I chose these variables to be the defining factors for difficulty due to simplicity (yet effective), and following conventions from other games within the infinite runner genre.

4.3.2 Road Movement

Levels work by means of the floor moving, with the player stationary. The 'floor' consists of 3 roads (bare minimum required - any less and it's obvious that part of the floor is teleporting), moving in a conveyor-belt fashion - with each road teleporting to the back of the conveyor once it reaches a certain point (past the player). To provide an overview of how the road movement works, is a diagram in figure 4.17:

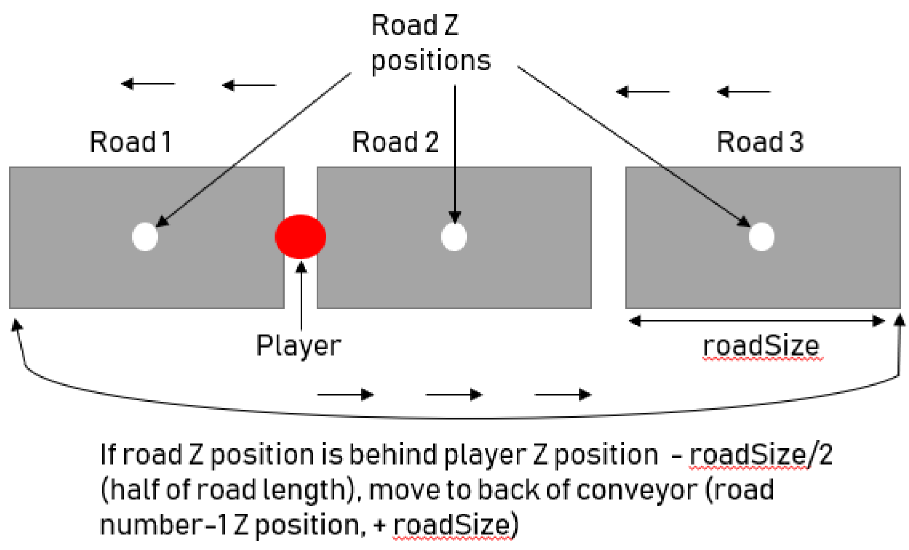

Road Z positions
Road 1
Road 2
Road 3
Player
roadSize
If road Z position is behind player Z position - roadSize/2 (half of road length), move to back of conveyor (road number-1 Z position, + roadSize)

FIGURE 4.17: A diagram showing how the road movement works

Each road has a static variable setting to decide which other road it is 'chained' onto the end of (var *road*), as shown below:

```
switch (this.name)
      {
          case "Road1":
              GameObject road3 = GameObject.Find("Road3");
              road = road3;
              break;
          case "Road2":
              GameObject road1 = GameObject.Find("Road1");
              road = road1;
              break;
          case "Road3":
              GameObject road2 = GameObject.Find("Road2");
              road = road2;
              break;
      }
```

FIGURE 4.18: The code that determines which part of the chain the current road is a part of

4.3.3 Obstacles and Power-ups

Spawning

Within the road object, various obstacles and power-up objects are procedurally generated. This is accomplished within MoveFloor.cs, with the following steps:

1. Random value generated based on difficulty levels

2. Every x seconds (based on value from step 1), a new object is spawned with spawnGen() function

The spawn gen function then decides which object to spawn based on weighted values, as per follows:

TABLE 4.1: Object spawns

% chance	Object	Type
25%	Fire hydrant	Obstacle
25%	Poop	Obstacle
30%	Manhole	Obstacle
10%	Dog bone	Power-up
10%	Energy drink	Power-up

These values have been chosen due to keeping the levels challenging - the obstacles should have a majority chance to spawn, else the level is just full of power-ups with no challenge (and thus no fun) at all.

Once an object has been chosen, a random coordinate based on the road itself is generated - so the object is spawned within the bounds of the road. The object is then instantiated and parented to the road; thus so it moves with it. This can be seen below in figure 4.19:

```
spawnX = Random.Range(16, 25);
          float zLower = this.transform.position.z - 25;
          float zUpper = this.transform.position.z + 25;
          spawnZ = Random.Range(zLower, zUpper);

          var obstacle = Instantiate(spawn, new Vector3(spawnX,
               0.5f, spawnZ), Quaternion.identity);
          obstacle.transform.parent = this.transform;
```

FIGURE 4.19: The code that determines where an object is spawned on the road

Behaviour

Object spawn behaviour is defined within SpawnBehaviour.cs. This script employs Unity's OnTriggerEnter function to tell when there has been a collision between the player and the spawn object. A switch statement is called to determine whether the player should lose a life, or gain a life - if the object name corresponds to an obstacle, a life is lost (and obstacle audio is played), and if the name corresponds to a power-up, a life is gained (unless lives < 5). A snippet of the complete statement can be seen below, in figure 4:

```
if (collision.gameObject.name == "LevelPlayer" || collision.
    gameObject.name == "Dog" && LevelPauseMenu.pause == false)
        {
            lives = LevelPlayerController.lives;
            switch (this.name)
            {
                case "firehydrant Variant(Clone)":
                    if (lives >= 1)
                    {
                        lives -= 1;
                        LevelPlayerController.
                            audioPlayObstacle = true;
                    }
                    break;
```

FIGURE 4.20: The code that determines what happens upon collision with an obstacle

Upon collision, the spawn object is destroyed to prevent double collisions (and thus losing/gaining multiple lives at once). Lives can also only be lost if the current lives are >= 1, as, even though the level over menu trigger upon 0 lives, a negative value could sometimes be obtained if a player collided with 2 different obstacles at the same.

Additionally, OnTriggerEnter rules when the spawn objects should be destroyed; upon collision with a 'killbox' object located behind the player within each level scene - seen in figure 4.21. Objects are destroyed after passing the player in order to prevent them from repeating the conveyor, and potentially adding into a mass.

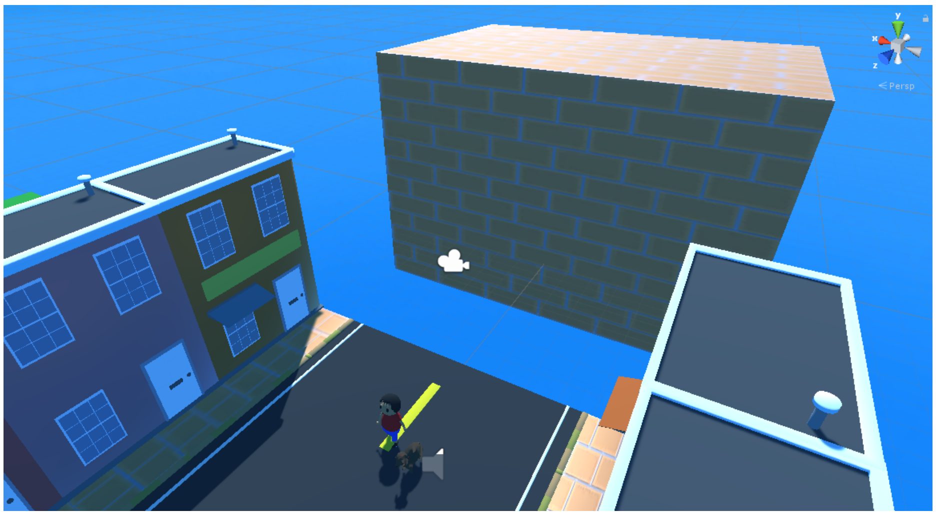


FIGURE 4.21: The 'killbox' that destroys objects once they pass the player

4.4 UI

Relevant scripts for this section: BulletinUI.cs, LevelOverMenu.cs, LevelOverUIText.cs, LevelPauseMenu.cs, LevelUIText.cs, PauseMenu.cs, StartMenu.cs

Interfaces can be viewed in section 5.4

The UIs within the project generally work as follows (with the exception of the start interface and level overlay):

- Canvas parent game object, with UI details as inactive children
- When condition for interface is met, toggle UI details active

4.4.1 Start Interface

The start interface is the first thing seen when the game is loaded. Containing buttons to Start and Quit the game, it follows a simple design as to not overload the player upon launch.

The startButton() function is applied to the start button UI element, in order for it to be called once the button is pressed. Once pressed, the timer variable and fade in animations are triggered, and the hub scene loading after a few seconds (allowing fade in animation to complete).

```
public void startButton()
    {
        fadeIn.SetActive(true);
        timerOn = true;
    }

 void Update()
    {
        if (timerOn == true)
        {
            timer += Time.deltaTime;
            if (timer >= 2.0f)
            {
                SceneManager.LoadScene("Hub", LoadSceneMode.
                    Single);
            }
        }

    }
```

FIGURE 4.22: Code displaying how the timer method works, within the start button method

The quitButton() function is applied to the quit button UI element, in order for it to be called once the button is pressed. Once pressed, the following code is executed in order to quit the application:

```
public void quitButton()
    {
        Application.Quit();
    }
```

FIGURE 4.23: Code that quits the application

4.4.2 Pause Interface

Details on how to toggle the pause interface on and off are discussed in section 4.1.2.

The resumeButton() function is applied to the resume button UI element, in order for it to be called once the button is pressed. Once pressed, the pause is toggled off, as seen in figure 4.4.

Hub pause only: Identical to the Start Interface in 4.4.1, the quitButton() function is applied to the quit button, and calls the code in figure 4.23 to quit the application.

Level pause only: The returnToHub() function is applied to the return to hub button, and calls the code in figure 4. Identical to the startButton() function in 4.4.1, a timer is triggered to allow the fade in animation to play 4.22 and load the hub scene.

4.4.3 Bulletin Board Interface

Once the bulletin UI is set active, as detailed in 4.1.3, the bulletin UI home is set active. From here, it can be navigated to read about the game, and the controls.

Below, in figure 4.24, is a flowchart to visualise how the setActive method works for this interface:

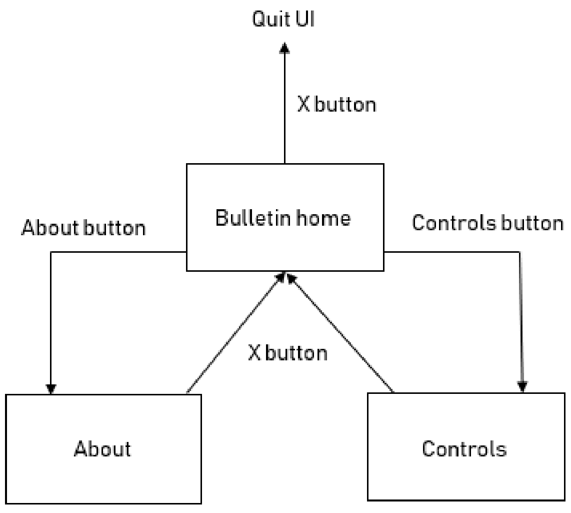


FIGURE 4.24: A flowchart displaying the flow of the bulletin interface

4.4.4 Level Over Interface

The level over interface is set active from LevelPlayerController.cs, once lives <= 0. Within this interface, the final score is taken from LevelPlayerController, and converted to string for display within the interface.

Nested if statements are utilised to decipher how many stars the player receives, if any. If the final score is over 200, the player gets one star, over 500 is two stars, and over 1000 is three stars (figure 4). Balancing was integral to get this component correct - involving a lot of playtesting across different levels to achieve what felt right.

```
score = LevelPlayerController.distanceInt; //gets the final
   score from the LevelPlayerController script
        if (score >= 200)
        {
            star1.SetActive(true); //if score is 200 or over,
                player gets 1 star
                if (score >= 500)
                {
                    star2.SetActive(true); //if score is 500
                       or over, player gets 2 stars
                        if (score >= 1000)
                        {
                            star3.SetActive(true); //if score
                                is 1000 or over, player gets
                               3 stars
                        }
                }
        }
```

FIGURE 4.25: Code that determines how many stars the player is awarded

Three stars is intended to be a challenge - easier to achieve on easy levels, and demanding on harder levels. This provides an appeal to players of varying experience - casual players can still feel a sense of accomplishment by winning 3 stars on easier levels, with more experienced players looking to test their ability in achieving 3 stars on hard (and very hard) levels.

The returnToHub() functions identically to the implementation described in section 4.4.2.

4.4.5 Level Overlay

The level overlay displays the score and lives on the screen during levels. These variables (lives and distance) are defined as public static variables, held within LevelPlayerController.cs, enabling LevelUIText.cs to access them. The variables need to be accessed in order to convert to string for application to the overlay text elements - raw float values cannot be used.

```
livesText.text = LevelPlayerController.lives.ToString
    ();
distanceText.text = LevelPlayerController.distanceInt
    .ToString() + "m";
```

FIGURE 4.26: Code that converts the lives and distance variables to strings - compatible with UI

Three was chosen as the number of lives due to typical use in games - being a fair number, not too high and not too low. Scoring (distance in meters) is updated in real time - rather than, for example, having to collect something to boost score - at an appropriate speed (equatable to speed of player) to allow the player to focus purely on avoiding obstacles and gaining power-ups.

4.5 Graphics

The overarching graphical theme of the game was to be colourful and cartoon-ish. I chose this theme as it's inline with other successful games of the same genre, in addition to appealing to a wider audience, and possible for me to create satisfactory models for.

4.5.1 Environment

Models

Environment models followed the specified theme, and all had appropriately sized box collider components added to them to prevent the player from walking through them. An example model, with collider, can be seen below, in figure 4.27:

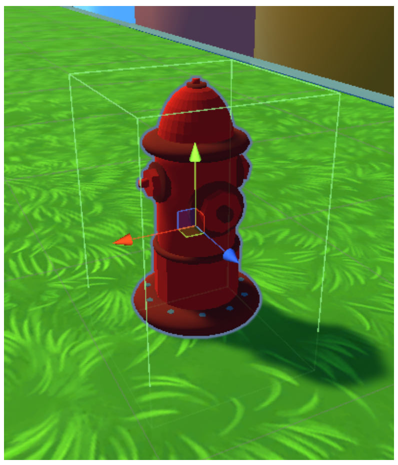


FIGURE 4.27: A fire hydrant with a collider component attached

Certain models, such as the dog and human models, and town and suburban houses, had differing colours applied to them (and prefab variants made) within Unity (rather than Blender) in order to provide aesthetic variation within the game.

Hub

The hub environment spans the town, park, and suburb area. A terrain was used to house the hub scene, allowing me to paint and raise various ground textures appropriate to the environment. Raising the ground textures also proved beneficial in closing off the map from the undeveloped areas of the game environment.

Models were placed appropriately (with scaled sizing) - with foliage placed on grassy areas, and models such as fire hydrants placed on pathways. The town area was designed around the intended experience for the player when first spawning in - the spawn point being the town hall (serves as a noticeable landmark within the hub). A green area was added just ahead, for not only aesthetic purposes (looks nicer to see both built-up and foliage, rather than just build-up), but to serve as a 'welcome area' for the player, housing the bulletin board.

Particle systems were utilised to make attention-grabbing objects (fast moving, gradient colours, woosh audio attached - invites the player to approach them), serving as the level indicators. Intuitive colour systems were added to them - green for easy, orange for medium, red for hard. Additionally, manholes serve as entrances to secret sewer levels as a result of playtesters expecting to be able to enter them (open model).

Levels

Levels had set dressing applied to them as appropriate: town levels had townhouses either side of the road, park levels had grass and trees, and suburb levels had suburban houses. This can be seen below in figure 4.28:

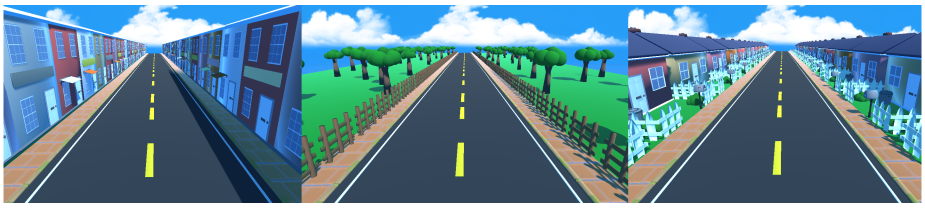


FIGURE 4.28: Comparison views between the level set dressing

4.5.2 Characters

Models

Models created for the project were a dog model, and a base human model. The human model was modified - by selecting faces and adjusting colour (for clothes and hair) or size (for hair length, female AI skirts) - to produce male and female AI, and the player variations.

The player's appearance is unique, wearing blue shorts, a red top, socks, and black shoes, rather than being the same as the male AI, in order to make the player stand out within the game. Male AI wear long trousers, and female AI wear skirts with pumps, as seen in figure 4.29 below:

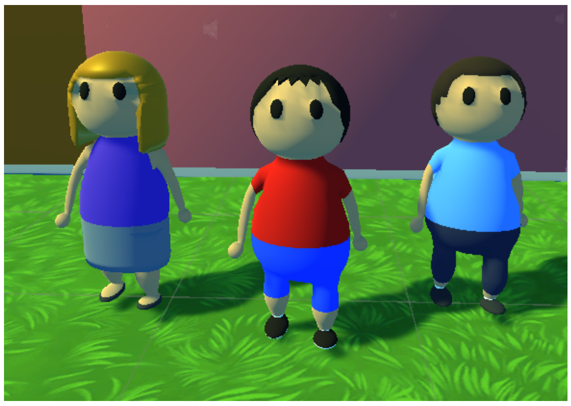


FIGURE 4.29: The three human models used in the project

Animation

Before animation could commence, models had to be prepared with rigging in order to thus pose and animate them.

For animation, I referred to various human and canine cycle videos for idling, walking, and running (for example, in figure 4.30), in order to attempt to create realistic-looking animations.

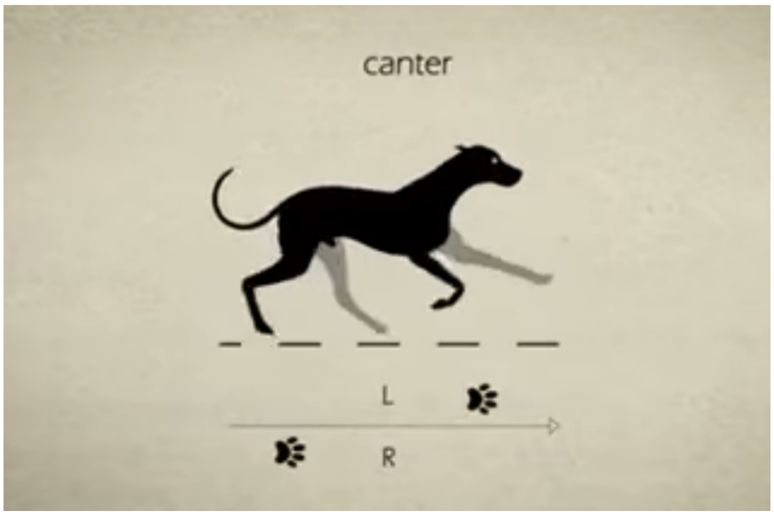


FIGURE 4.30: An example of an animation cycle reference (YouTube, 2017)

Animator controllers (as seen in section 4.1.1, figure 4.3) were created and applied to dog and human AIs, the player, and trees (swaying in the breeze to provide a more detailed environmental atmosphere).

The default animation for all is idle, with RunOn and WalkOn parameters changing to 1 when the AI or player move - and back to 0 (idle) when they stop moving.

4.6 Scrapped Features

Unfortunately, due to time constraints, some of the originally planned features that were less integral to the project had to be scrapped.

4.6.1 Saving

The save feature was designed to hold the players highest score at the end of each level. This would have involved storing the first score value for each level to a variable, and saving it using Unity's PlayerPrefs function (Unity, 2019). Subsequent attempts at the same level would have checked if the final score was more than the saved value, and if so, overwriting it.

The scores would have been able to have been viewed on the bulletin board, along with the corresponding star value. Justification for design of this feature comes from encouraging replayability of the game, with the player being able to see their high score and attempt to beat it by replaying the levels.

4.6.2 Reward System

A reward system in the game was a large feature planned to take the form of a hat shop, where players could exchange stars earned from levels for cosmetic hats.

The shop would have either been simply an AI standing outside, or inside a building - if the player walked to the door of the shop within the hub, and held down space, they would be teleported to this area.

The shop interface would show pictures of each hat (e.g. crown, baseball cap, jester hat), and their price. Prices would have either come from earning stars cumulatively - allowing players to earn all rewards from any level - or from having to earn each star once from every level - encouraging the player to experience every level in the game.

Once a hat is 'bought', the original player model would be set inactive, and the corresponding hat model set active.

An example of a hat model I created can be seen below, in figure 4.31:

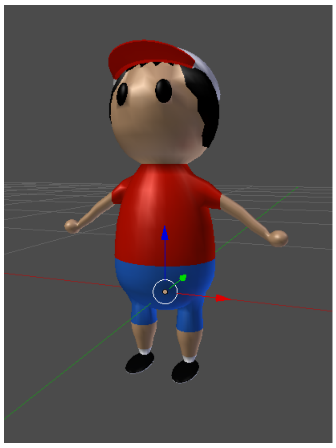


FIGURE 4.31: A model of the player wearing a baseball cap

Like the save feature, the reward system would have also boosted the games replayability value - giving the player something to work forward to - as well as providing the player with a feeling of achievement.

4.6.3 AI Responses

AI responses was a last-minute hopeful addition to the game, in order to add more life and character to the AI in form of player-AI interaction. The AI were planned to say various statements in form of a speech bubble object (that is their child) activating upon collision, such as ‘Hello’ on first collision, and ‘Go away’ on the 10th collision.

Chapter 5

User Guide

5.1 Technical Specifications

The game runs on Windows and Mac operating systems.

1. Download the game folder
2. Extract files
3. Run 'Walkies.exe'
4. Click play!

5.2 User Controls

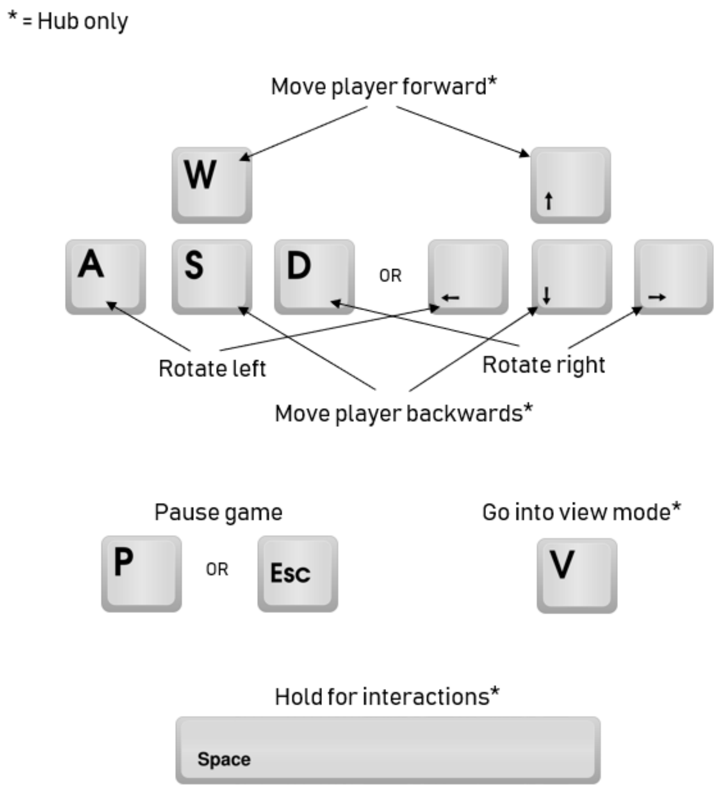


FIGURE 5.1: Controls for Walkies (WPClipart, 2019)

5.3 Gameplay

5.3.1 Hub Gameplay

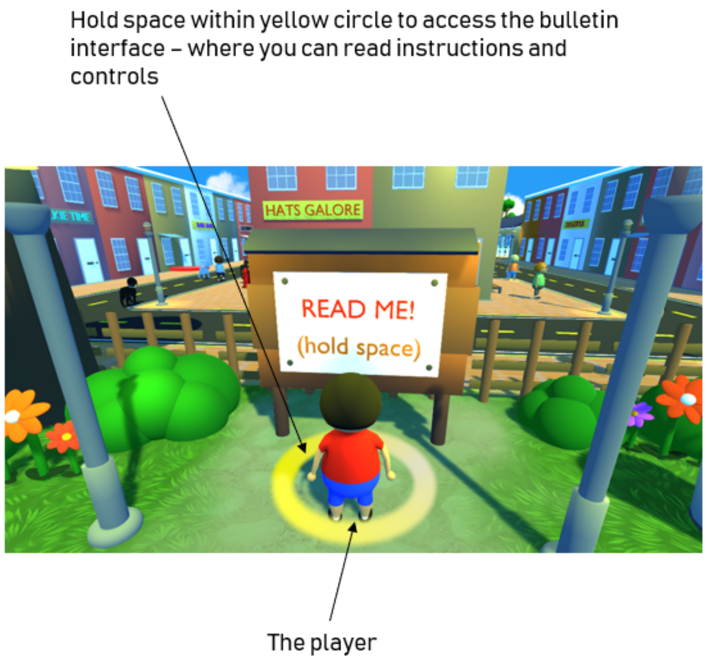

Hold space within yellow circle to access the bulletin interface – where you can read instructions and controls
HATS GALORE
READ ME!
(hold space)
The player

FIGURE 5.2: Accessing the bulletin board

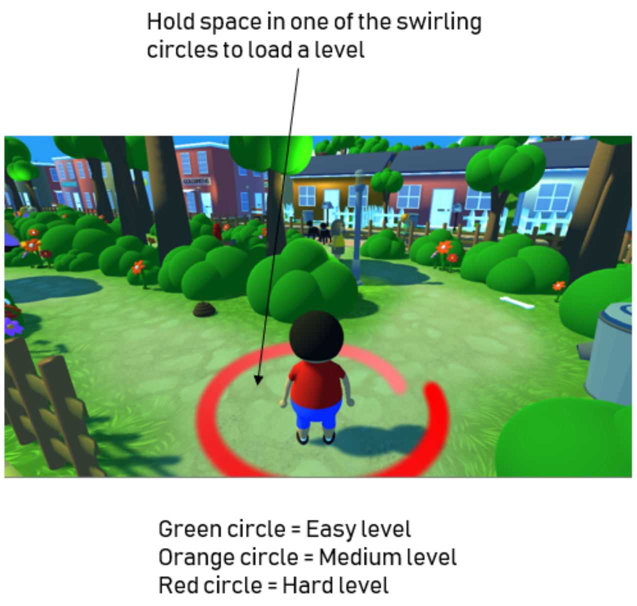

Hold space in one of the swirling circles to load a level
Green circle = Easy level
Orange circle = Medium level
Red circle = Hard level

FIGURE 5.3: How to access levels

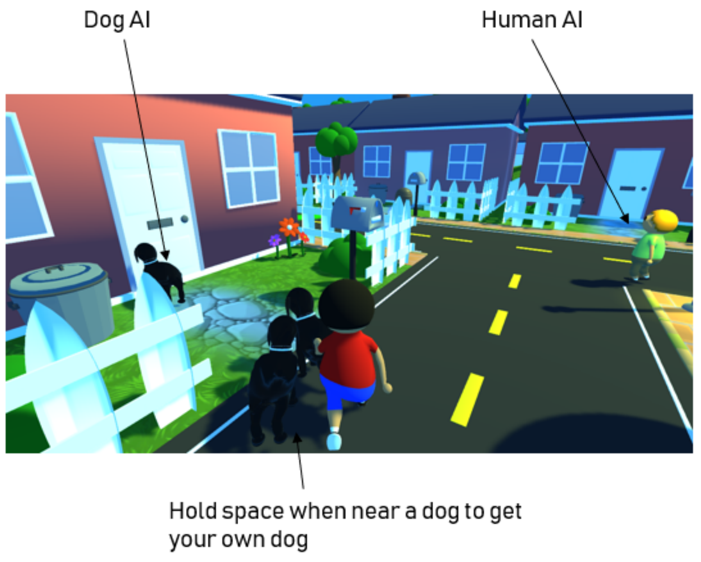


FIGURE 5.4: How to get a dog companion

5.3.2 Level Gameplay

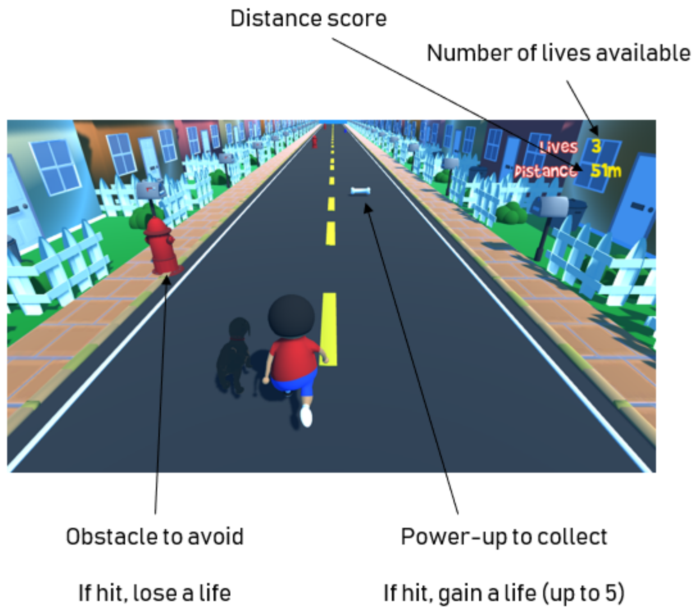


FIGURE 5.5: Level gameplay

5.4 Interface

5.4.1 Start Interface

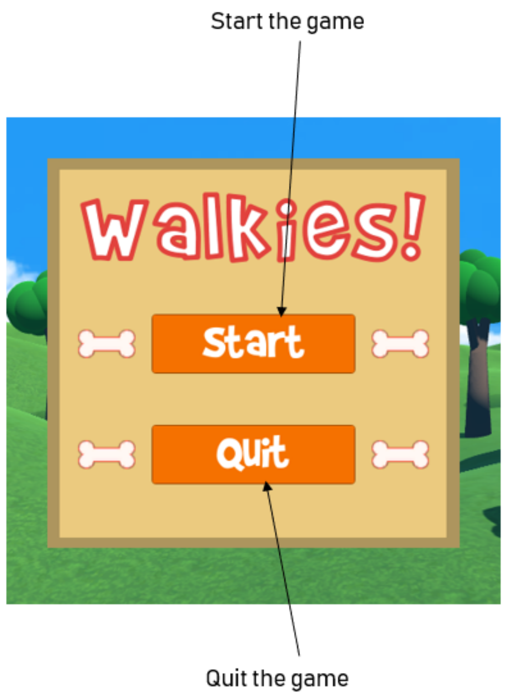


FIGURE 5.6: Start menu

5.4.2 Pause Interface

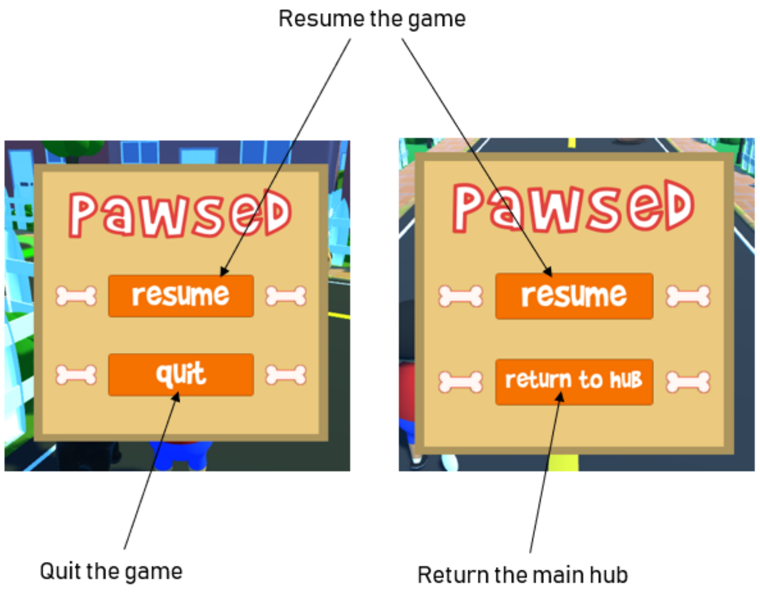


FIGURE 5.7: Pause menus; hub on the left, levels on the right

5.4.3 Bulletin Interface

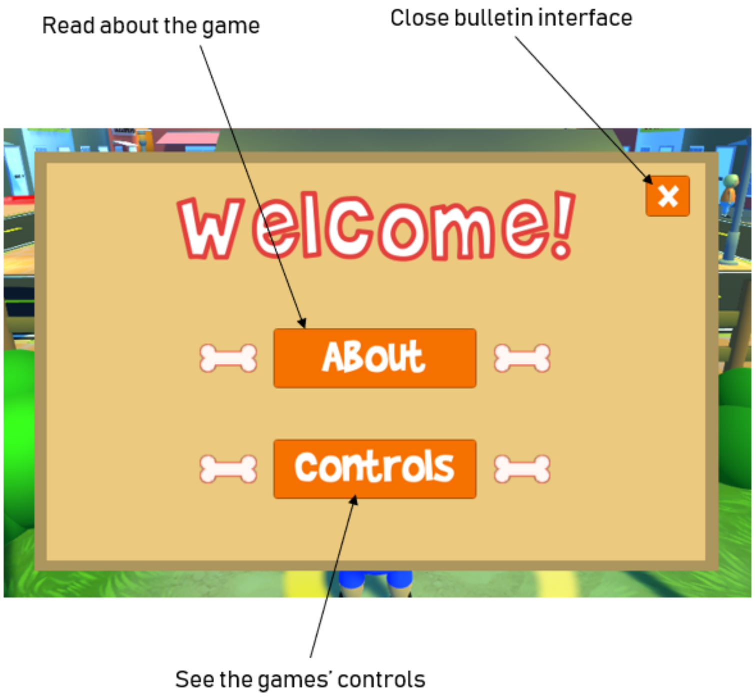
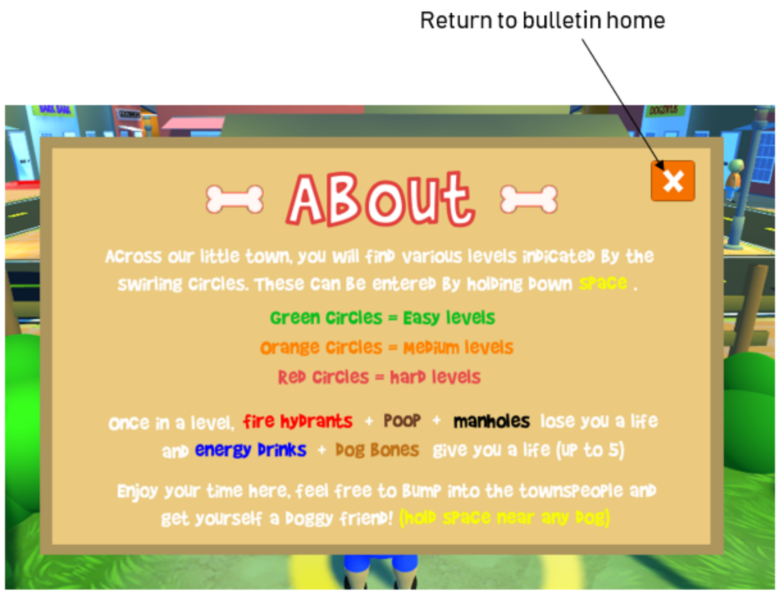


FIGURE 5.8: Bulletin board home interface



About

Across our little town, you will find various levels indicated by the swirling circles. These can be entered by holding down space .

Green Circles = Easy levels

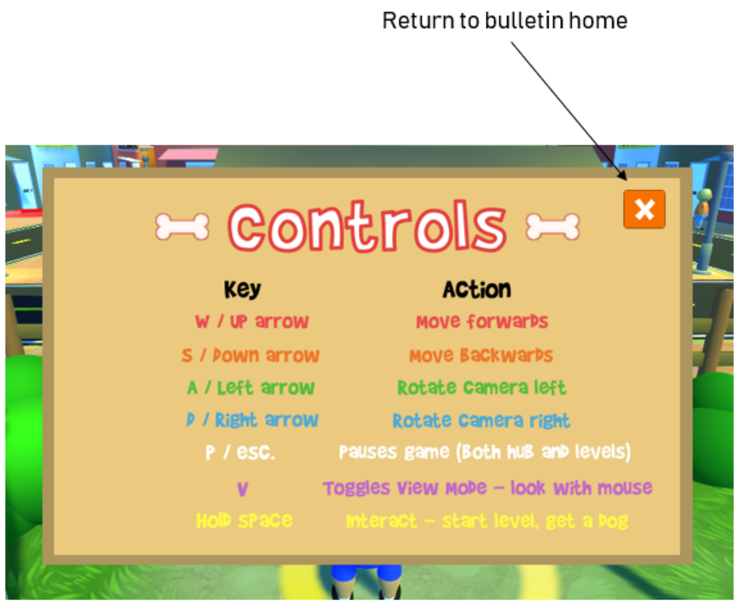
Orange Circles = Medium levels

Red Circles = hard levels

Once in a level, fire hydrants + Poop + manholes lose you a life and energy Drinks + Dog Bones give you a life (up to 5)

Enjoy your time here, feel free to bump into the townspeople and get yourself a Doggy friend! (hold space near any Dog)

FIGURE 5.9: Bulletin board about interface



Controls

Key	Action
W / Up arrow	Move forwards
S / Down arrow	Move Backwards
A / Left arrow	Rotate Camera left
D / Right arrow	Rotate Camera right
P / esc.	Pauses game (Both hub and levels)
V	Toggles View Mode – look with mouse
Hold space	Interact – start level, get a Dog

FIGURE 5.10: Bulletin board controls interface

5.4.4 Level Over Interface

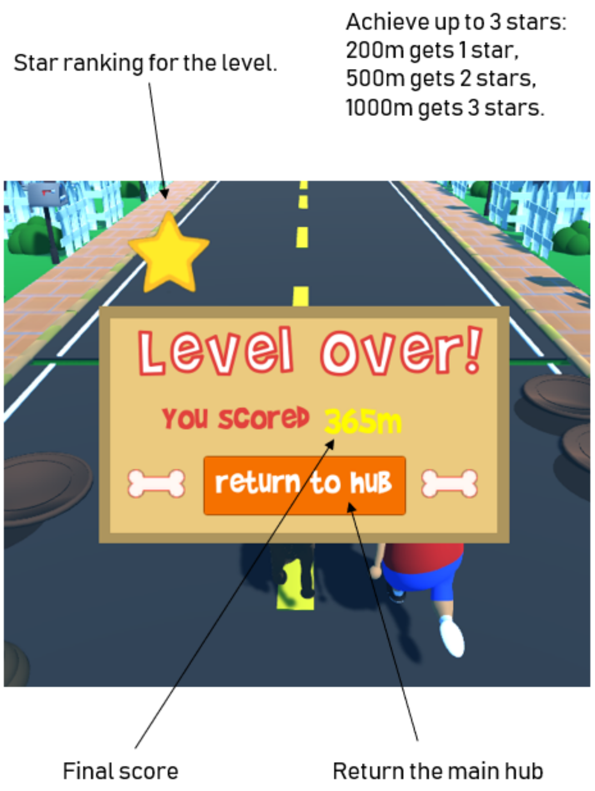


FIGURE 5.11: Level over menu

Chapter 6

Testing

For testing, I carried out both white-box and black-box testing, guided by specifications set out in Chapter 3. The test cases don't cover every specific feature in great detail; generally these were tested along the development process by myself. Instead, these cases cover the general and most important objectives necessary to have the project functioning.

White-box testing refers to being knowledgeable of the internal structure of the software - and thus I carried this out.

Black-box testing is the opposite - with the internal structure unknown to the user - and so playtesters carried this out for me, providing me with results.

These test cases are in addition to constant testing I performed throughout development of the project. Methods I used to help with this constant testing included colouring objects differently (to easily differentiate and thus see if they were working as intended), and printing statements in certain parts of the code to check if the correct code was executed.

6.1 White-box Testing

TABLE 6.1: Test Case 1: Player Controls

Feature	Player controls (movement etc.)
Action	Input all of the player controls
Process	Player is moved, or UI is toggled
Expected Output	Player moves forward/backwards, pause toggles on/off, view mode toggles on/off
Output	Player moves forward/backwards, pause toggles on/off, view mode toggles on/off
Result	Pass

TABLE 6.2: Test Case 2: Level Load (for all levels)

Feature	Level loads from hub scene
Action	Hold down space within radius of level indicator
Process	Correct difficulty and level setting are passed as values
Expected Output	Corresponding level loads (correct setting, and difficulty)
Output	Corresponding level loads
Result	Pass

TABLE 6.3: Test Case 3: Road Moving

Feature	Road moves within levels
Action	N/A
Process	Road constantly moving, checks current position, teleports when behind player to back
Expected Output	Road moving in a believable way (player looks to be moving instead)
Output	Road moving in a mostly believable way (player looks to be moving instead) - slight visual gaps in between roads
Result	Mostly pass

TABLE 6.4: Test Case 4: Spawn Collisions

Feature	Obstacles and Power-ups collide with player
Action	Run into obstacle/power-up
Process	Object detect player, loses/gains a life dependent on name
Expected Output	Player loses/gains a life, obstacle disappears
Output	Player loses/gains a life, obstacle disappears
Result	Pass

TABLE 6.5: Test Case 5: Dog Companion

Feature	Gaining a dog companion in the hub scene
Action	Hold down space whilst near a dog AI
Process	Dog of same colour is set active
Expected Output	Dog of same colour appears by player
Output	Dog of same colour appeared
Result	Pass

6.2 Black-box Testing

TABLE 6.6: Test Case 1: Graphical Requirements

Feature	Graphical aspects
Action	Explore the game
Expected Results	Consistent, pleasing theme, all models working, no complaints
Results	Consistent, pleasing theme, dog has see-through model, some models could do with more variation
Result	Mostly pass

TABLE 6.7: Test Case 2: Appropriate Balancing

Feature	Overall balancing
Action	Play the game, specifically levels
Expected Results	Easy felt easy, hard felt challenging but achieveable with effort
Results	Easy felt easy, hard felt challenging but achieveable with effort
Result	Pass

TABLE 6.8: Test Case 3: Intuitive UI

Feature	Interfaces
Action	Interact with the various interfaces
Expected Output	Easy to use, no external help needed with playing the game
Results	Easy to use, no external help needed with playing the game
Result	Pass

6.2.1 Conclusion

To briefly conclude this section, both the white and black-box testing suggests my project is successful - although it's not perfect as a few minor imperfections were found, the vast majority of tests succeeded with a pass. These results imply the project is both functional, and enjoyable as set out to be in my original aims.

Chapter 7

Evaluation

7.1 User Evaluation

7.1.1 User Feedback

I carried out a survey with a few select playtesters, in order to gain both qualitative and quantitative feedback on my project. The full survey can be found in Appendix C.

Half of the participants were of age 18-21, and half age 27+. 60% were male, and 40% female - providing a good variation of users to gain feedback from. More importantly, people of a wide variety of gaming background took the survey - important to gauge whether the game is accessible to everyone (as is typical of infinite runner genres) and thus evaluate stakeholders.

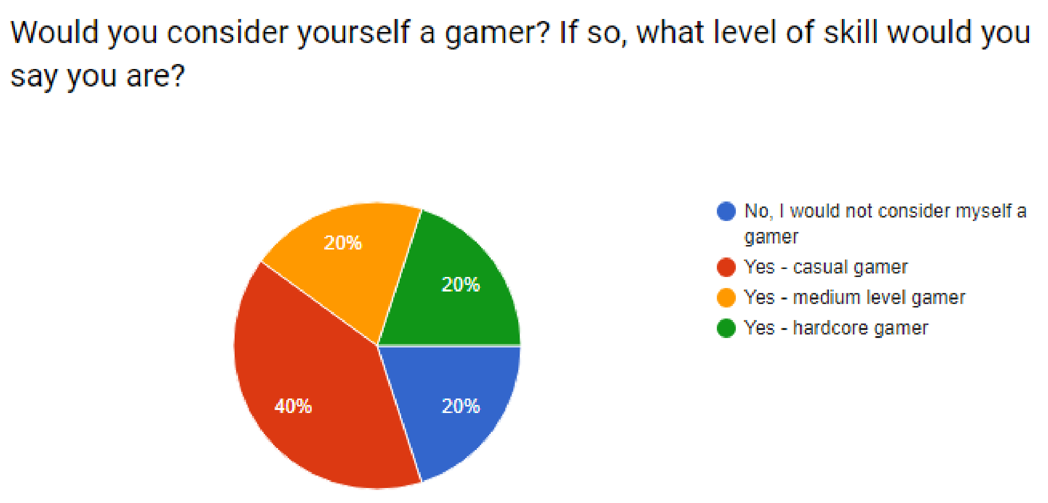
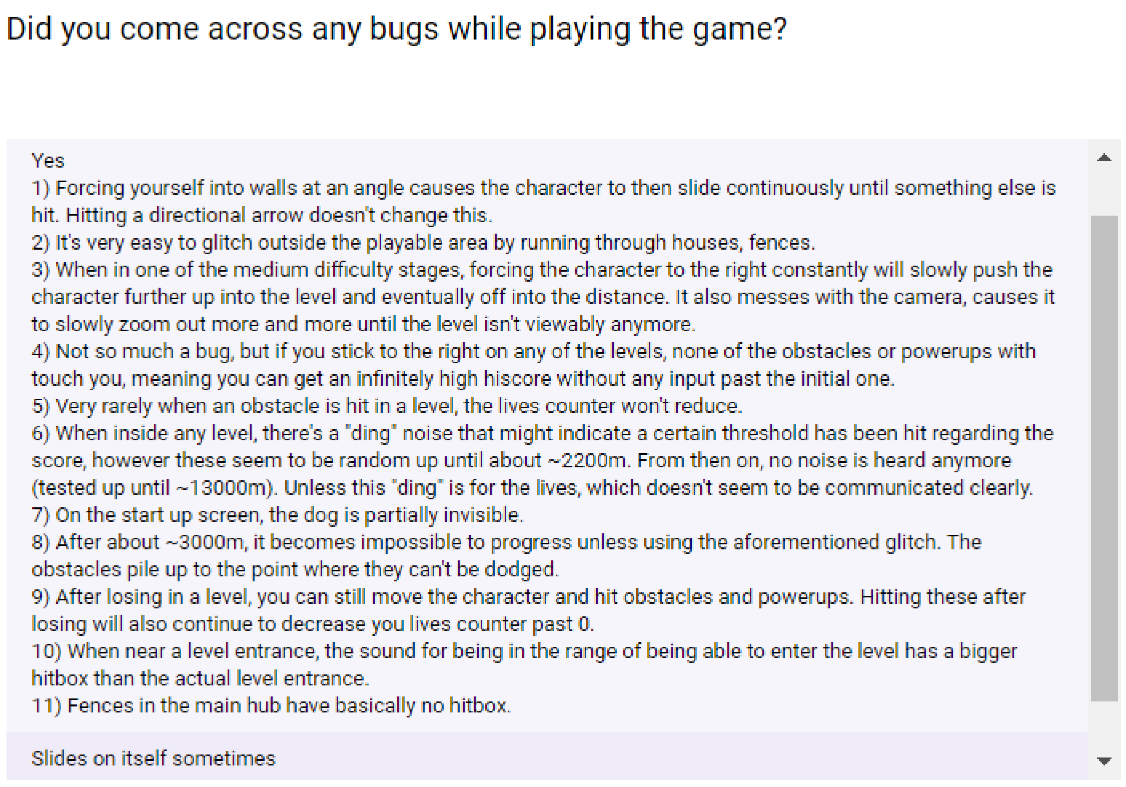


FIGURE 7.1: Gaming experience of people who took the survey

Multiple players provided graphical feedback - to be expected, as I am new to creating models and animations. This included the dog model being see-through, and dog feet looking weird, the models looking too similar and basic etc. I'm not concerned about this feedback particularly - these are all quite specific points, rather than generally stating that the game doesn't look good. This projects main focus is on technical specifications, rather than the graphical aspects. Despite this, I do acknowledge graphics as a point of improvement for the game.

Various bugs were reported throughout development, as seen in figure 7.2 below. Evaluating the game now as a finished product, I'd argue its success due to the fact that the vast majority of bugs reported in earlier development gameplay have since been fixed (detailed further in section 7.2).


Did you come across any bugs while playing the game?

Yes
1) Forcing yourself into walls at an angle causes the character to then slide continuously until something else is hit. Hitting a directional arrow doesn't change this.
2) It's very easy to glitch outside the playable area by running through houses, fences.
3) When in one of the medium difficulty stages, forcing the character to the right constantly will slowly push the character further up into the level and eventually off into the distance. It also messes with the camera, causes it to slowly zoom out more and more until the level isn't viewably anymore.
4) Not so much a bug, but if you stick to the right on any of the levels, none of the obstacles or powerups with touch you, meaning you can get an infinitely high hiscore without any input past the initial one.
5) Very rarely when an obstacle is hit in a level, the lives counter won't reduce.
6) When inside any level, there's a "ding" noise that might indicate a certain threshold has been hit regarding the score, however these seem to be random up until about ~2200m. From then on, no noise is heard anymore (tested up until ~13000m). Unless this "ding" is for the lives, which doesn't seem to be communicated clearly.
7) On the start up screen, the dog is partially invisible.
8) After about ~3000m, it becomes impossible to progress unless using the aforementioned glitch. The obstacles pile up to the point where they can't be dodged.
9) After losing in a level, you can still move the character and hit obstacles and powerups. Hitting these after losing will also continue to decrease you lives counter past 0.
10) When near a level entrance, the sound for being in the range of being able to enter the level has a bigger hitbox than the actual level entrance.
11) Fences in the main hub have basically no hitbox.

Slides on itself sometimes

FIGURE 7.2: Reported bugs within the game

Although game optimisation wasn't a particular specific goal in making this game, it is of course a good goal to meet. The majority of playtesters reported smooth gameplay - although some reported lag, this is arguably due to playing the game on older, or non-gaming computers, and thus isn't necessarily inherently a problem with the project itself.

The majority of playtesters also reported finding the controls/objectives of the game to be clear - with a minority stating 'partly clear'. As player feedback was implemented into the game after this survey was taken, it's highly likely that the same players who answered 'partly clear' would now answer 'Yes - clear' on the final version.

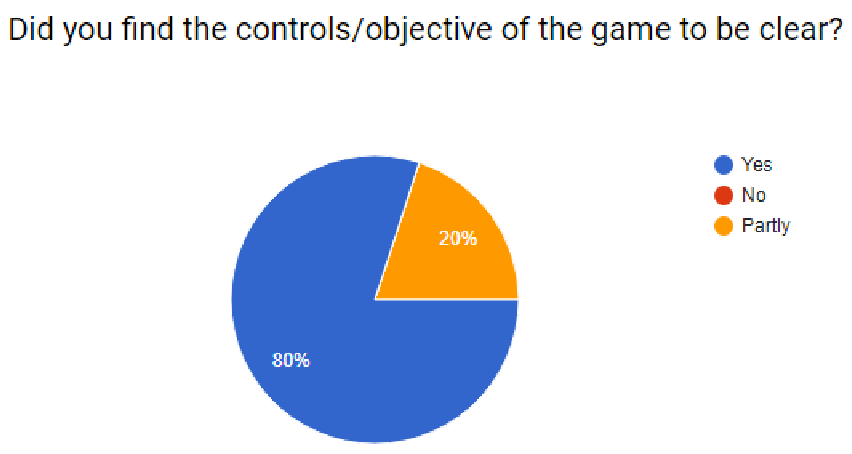


FIGURE 7.3: Were controls and objectives clear?

Respondents didn't have much to say about the UI, expect for one player who stated 'Good variety of colours, nice large images, overall looked good' - impling the UI design overall is a success.

In terms of balancing. feedback varied from player to player - hence why balancing is so difficult to get right. Some players stated they felt 3 stars was too much of a challenge, some stated it was too easy. These point suggest I could further tweak the balancing, although a different feature to fix these problems may be preferential instead; such as generate the levels based on the players' stated gaming experience.

The majority of players did not find the secret levels (which were implemented as a result of player request for an easter egg, and stating that it appears you should be able to go down the manholes), with only 20% finding them. This is good, as it means the levels do what their name describes - be secret!

Other various comments made included a larger variety of audio tracks (repetitive to listen to), and dialogue with the townspeople. I feel this feedback is great to get, as it's not commenting on the core gameplay aspects; thus I can only presume that the core objectives have been met successfully.

7.1.2 Improvements

As a result of various feedback from multiple playtesters - both from the survey results, and from conversational feedback - here are features I implemented into the final project version:

- Added WASD for player movements (majority of players stated they prefer WASD to arrow keys on the survey)
- Gave the player the ability to move backwards
- Added the Escape key to pause
- Gave the player the ability to gain a dog companion
- Added a killzone to prevent out-of-bounds access

- Added more colliders to prevent out-of bounds access
- Provided more specific information on the bulletin UI (specified maximum of 5 lives, obstacles would lose/gain the player a life)
- Froze player coordinates to prevent unintended float across axis
- Expanded the object spawn area within levels to reach some safe spots
- Added secret levels in the sewers, accessible via manholes

This is quite a substantial list, and perfectly shows how valuable playtesting a game is. While none of these changes or additions took particular skill to implement, they have arguably positively impacted players' experience of the game.

7.2 Specification Evaluation

7.2.1 Gameplay requirements

Enjoyable to play?

While this point is rather subjective, my experience watching playtesters try out the game, and myself trying out the game, suggests this point has been met - I certainly enjoyed myself attemping to gain 3 stars throughout the levels.

Ability to explore hub scene, with appropriate collisions and interactions?

The objective has been met, with a fully-working hub scene, containing interactions for the level indicators, dog companions, and bulletin interface. All collisions work as intended, with players being unable to clip through models.

Ability to access infinite runner levels from various points in the hub scene?

This objective has been met, with players able to access the infinite run levels from various points within the hub.

Infinite runner levels with moving roads, randomly spawned obstacles or power-ups, life system, and score system?

This objective has been met, with correctly functioning moving roads, spawned obstacles and power-ups, life system, and scoring system.

Appropriate balancing - in terms of level difficulty, overall player control sensitivity/speed, and AI action weighting?

Arguably, this objective is met. Whilst it's hard to determine the accuracy of balancing, feedback and personal experience suggests balancing is at least fine, if not good. At the very least, all balancing bar level difficulty has been successful.

Reward system - hat shop, ability to purchase hats for stars?

Unfortunately, this is the largest requirement that has not been met within the gameplay objectives, purely due to time constraints. Fortunately, this is not a core aspect - the game can function without it, however it definitely would have been a valuable feature to implement (in terms of boosting replay-value and awarding the player)

7.2.2 AI requirements

Ability to perform a variety of actions: idling, walking, running,and flocking?

This objective has been met, with both the dog and human AI able to perform idling, walking, running, and flocking behaviours.

Ability to perform actions in a weighted fashion?

This objective has been met, with both the dog and human AI able to perform their actions in a weighted manner - the weighting makes sense in terms of providing natural gameplay, as justified in Chapter 4.

Ability to gain a dog that follows the player upon interacting with the dog AI?

This objective has been met, and improved as a result of player feedback - specifically stating that there should be a choice of colour available for the dog (which has been implemented - brown, black, white, grey, and blonde dogs available). The player can gain a dog companion by pressing spacebar when near a dog AI.

Ability to express discontent upon collision with the player?

This objective has not been met. Whilst unfortunate, this is the only objective within the AI requirements that has not been met, and it not integral to the AI functioning. However, as a player commented on the wish to be able to interact with the AI, this suggests this objective is more desired than I originally thought.

7.2.3 UI requirements

Overall consistent, thematic, and stylistic design?

This objective has been met, with player feedback commenting positively on the aesthetically pleasing UI design. All of the interfaces are consistent, with the same colour schemes, buttons, sizing, and font used. The style matches with that of the overall game.

Start menu - capable of starting or quitting the game?

This objective has been met, with the start menu capable of both starting and quitting the game upon initially loading the application.

Pause menu - capable of pausing game from hub or levels, and quitting the game?

This objective has been met, with both the hub and level scenes able to pause the game from any point - with no interference from other features (such as view mode, or moving whilst the UI is open).

Bulletin menu - contains details about controls and gameplay to inform the player?

This objective has been met - partly as a result of implementing player feedback. The bulletin contains relevant information on the games' instructions, as well as the controls.

Level UI - capable of displaying updated lives and score

This objective has been met, with the level overlay UI displaying the current amount of lives the player has, and their current distance score in meters.

7.2.4 Graphical requirements

Overall consistent, thematic, and stylistic design?

This objective has arguably been met, with all game models sharing the same colour palette, sizing, and style. The textures, skybox, and overall game environment additionally follow this stylistic theme - colourful and cartoony.

Appropriate sizing and placement of models?

This objective has mostly been met, with the majority of models appropriately sized and placed around the scenes. One minor point I would make, is that the suburban house (possible townhouses too) might be slightly too big in comparison to the player - the mailbox is taller than the player itself. However, they do blend in nicely to their surroundings, and thus this scaling issue may be better off as a stylistic choice.

Believable walk, run, and idle animation cycles?

This objective has also mostly been met, with a minor point about the dog walk animation being the inconsistency with its feet. This is only a minor point, however, with the rest of the walk, run and idle animations having a believable appearance.

7.3 Conclusion

In relation to the original proposal for this project, the results have suggested that this project is a success - with players being capable of playing the game completely (thus functionality requirements are met), and enjoying themselves - thus thus fun factor has also been met. Additionally, the vast majority of specification requirements were met, implying the project has been successful - although not perfect. There were a few objectives either partly met, or not met at all; this doesn't mean the project is a failure, as the core aspects were met, however they are definitely good points for improvements to consider in any future development.

At the most basic level, my original aim of making a polished, 3D infinite runner-style game have been met. Additionally I succeeded in meeting the goal of creating the majority of the assets myself; having made all of the models used in game, and animations.

Chapter 8

Conclusion

8.1 Overview

Primary objectives of this project consisted of creating a fun, consistently styled, infinite runner game. My project easily met all of these core objectives, and additional ones. Unfortunately, some aims were not met - such as implementing a reward system - however I do not believe this makes the project a failure due to the fact that the missed aims were optional - the core objectives were met without them.

It is hard to compare it to games of the same genre, due to the fact that they have been developed for different platforms (mobile), however, I feel as though my game is certainly an original twist on this genre (due to having a hub world, and AI etc.), and provides fun gameplay.

Overall, with gameplay, AI, UI, and graphical requirements mostly met, I would personally consider my game to be successful. However, of course, it can be difficult to determine a more accurate rating without commercially releasing the game; positive player feedback reaffirmed my opinion that this project is a success.

8.2 Future Work

Walkies has a vast number of possibilities available in relation to future development. First and foremost, scrapped features as outlined in section 4.6 would take priority when deciding on how to expand the game. To summarise, there are various - some previously mentioned, some not - additions that could be added, including:

- Save feature - detailed in 4.6.1
- Reward system - detailed in 4.6.2
- AI responses - detailed in 4.6.3. Additionally, this opens up the door for discussion regarding further, and more advanced, AI interaction, such as having conversations with selectable options and more.
- Environment atmospheric affects - this would include features like a day and night cycle, weather, and seasons
- Graphical improvements - with no time constraints, this allows for the possibility of re-making assets and animations to a higher quality
- More areas - the nature of the gameplay makes it very flexible in regards to settings, and thus more areas around the map could be opened, with more levels - such as a beach area, or forest area

- Level tweaks - more power-ups and obstacles added (some unique to certain levels), along with added effects for power-ups; such as a time-slower, score boost, invincibility etc.

8.3 Final Comments

This project has helped broaden my knowledge significantly in terms of my understanding of game development, and especially my knowledge of Unity and Blender - meeting objectives set out at the beginning of the project.

Ultimately, I am pleased with how the project turned out - I managed to meet the vast majority of specifications, specifically the core aims necessary to have a working game, and received positive user feedback, thus would consider this project a success overall.

References

Report

Business Insider. 2019. 'Grand Theft Auto 5' is still a best-selling game, over 5 years later - Business Insider . [ONLINE] Available at: https://www.businessinsider.com/grand-theft-auto-5-sales-2018-2019-1?r=US&IR=T. [Accessed 17 May 2019].

Eurogamer. 2014. The Last of Us review. [ONLINE] Available at: https://www.eurogamer.net/articles/2014-07-28-the-last-of-us-review. [Accessed 17 May 2019].

GameSpot. 2017. Star Wars Battlefront 2's Loot Box Controversy Explained - GameSpot. [ONLINE] Available at:
https://www.gamespot.com/articles/star-wars-battlefront-2s-loot-box-controversy-expl/1100-6455155/. [Accessed 17 May 2019].

Game Guides. 2019. Endings and branching of main quests - Fallout 4 Game Guide & Walkthrough | gamepressure.com. [ONLINE] Available at: https://guides.gamepressure.com/fallout4/guide.asp?ID=32694. [Accessed 17 May 2019].

Game Rant. 2016. No Man's Sky Steam Reviews Plummet to 'Overwhelmingly Negative' – Game Rant. [ONLINE] Available at: https://gamerant.com/no-mans-sky-steam-reviews-overwhelmingly-negative/. [Accessed 17 May 2019].

IGN. 2007. Super Mario Galaxy Review - IGN. [ONLINE] Available at: https://uk.ign.com/articles/2007/11/08/super-mario-galaxy-review. [Accessed 17 May 2019].

Jagex. 2018. Mining and Smithing Beta - News - RuneScape. [ONLINE] Available at: https://secure.runescape.com/m=news/mining-and-smithing-beta?gsi=bv2hhg. [Accessed 17 May 2019].

Kirk Hamilton. 2017. In-Game Purchases Poison The Well. [ONLINE] Available at: https://kotaku.com/in-game-purchases-poison-the-well-1820844066. [Accessed 17 May 2019].

Metacritic. 2019. Fallout 76 for PC Reviews - Metacritic. [ONLINE] Available at: https://www.metacritic.com/game/pc/fallout-76. [Accessed 17 May 2019].

Metacritic. 2019. Super Mario Galaxy for Wii Reviews - Metacritic. [ONLINE] Available at: https://www.metacritic.com/game/wii/super-mario-galaxy. [Accessed 17 May 2019].

Patricia Hernandez. 2015. Why Five Nights at Freddy's Is So Popular. [ONLINE] Available at:
https://kotaku.com/why-five-nights-at-freddys-is-so-popular-explained-1684275687.
[Accessed 17 May 2019].

Pocket Gamer. 2015. I Am Bread's controls may be too crumby for some. [ONLINE] Available at:
https://www.pocketgamer.com/articles/067324/i-am-breads-controls-may-be-too-crumby-for-som
[Accessed 17 May 2019].

RuneScape Wiki. 2019. Treasure Hunter - The RuneScape Wiki. [ONLINE] Available at: https://runescape.wiki/w/Treasure_Hunter#Controversies. [Accessed 17 May 2019].

Unity Technologies. 2019. Unity - Scripting API: PlayerPrefs. [ONLINE] Available at: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html. [Accessed 17 May 2019].

Wikipedia. 2019. Imangi Studios - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Imangi_Studios. [Accessed 17 May 2019].

Wikipedia. 2019. Kiloo - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/Kiloo. [Accessed 17 May 2019].

Wikipedia. 2019. List of best-selling game consoles - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/List_of_best-selling_game_consoles.
[Accessed 17 May 2019].

Wikipedia. 2019. List of best-selling video games - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/List_of_best-selling_video_games. [Accessed 17 May 2019].

Wikipedia. 2019. List of most-played mobile games by player count - Wikipedia. [ONLINE] Available at: https://en.wikipedia.org/wiki/List_of_most-played_mobile_games_by_player_count. [Accessed 17 May 2019].

Figures

Ars Technica. 2014. Ubisoft working to fix game-breaking Assassin's Creed: Unity bugs | Ars Technica. [ONLINE] Available at: https://arstechnica.com/gaming/2014/11/ubisoft-working-to-fix-game-breaking-assassins-creed-unity-bugs/.
[Accessed 17 May 2019].

COMPUTER / KEYBOARD KEYS - Public Domain clip art at WPClipart (image thumbnails page). [ONLINE] Available at: https://www.wpclipart.com/computer/keyboard_keys/. [Accessed 17 May 2019].

Goomba Stomp. 2016. Hall of Fame #11: 'Super Mario Galaxy' | Goomba Stomp. [ONLINE] Available at: https://www.goombastomp.com/hall-fame-super-mario-galaxy/.
[Accessed 17 May 2019].

YouTube. 2014. Candy Crush Saga Android Gameplay #14 - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=d5Rf0An-jEg. [Accessed 17 May 2019].

YouTube. 2018. DETROIT : BECOME HUMAN - MOCAP - MAKING OF - BEHIND THE SCENE - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=6TWHK18_ypA. [Accessed 17 May 2019].

YouTube. 2017. Dog WalkCycle and RunCycle Reference - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=8qV5pdO_X8U. [Accessed 17 May 2019].

YouTube. 2016. Goat Simulator ragdoll funny - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=CPlRaZNtZ3k. [Accessed 17 May 2019].

Project Assets

Audio

Freesound. (2019). Background Music Treatment by carpuzi. [ONLINE] Available at: https://freesound.org/people/carpuzi/sounds/382327/ [Accessed 17 May 2019].

Freesound. 2019. Concrete Footstep 1.wav by morganpurkis. [ONLINE] Available at: https://freesound.org/people/morganpurkis/sounds/384635/. [Accessed 17 May 2019].

Freesound. 2019. concrete footstep 1 by Yoyodaman234. [ONLINE] Available at: https://freesound.org/people/Yoyodaman234/sounds/166509/. [Accessed 17 May 2019].

Freesound. 2019. Dog bark 1 by jorickhoofd. [ONLINE] Available at: https://freesound.org/people/jorickhoofd/sounds/160092/. [Accessed 17 May 2019].

Freesound. 2019. Game background Music loop short by yummie. [ONLINE] Available at: https://freesound.org/people/yummie/sounds/410574/. [Accessed 17 May 2019].

Freesound. 2019. Game Start by plasterbrain. [ONLINE] Available at: https://freesound.org/people/plasterbrain/sounds/243020/. [Accessed 17 May 2019].

Freesound. 2019. game teleport by Leszek_Szary. [ONLINE] Available at: https://freesound.org/people/Leszek_Szary/sounds/133279/. [Accessed 17 May 2019].

Freesound. 2019. Lose_C_07 by cabled_mess. [ONLINE] Available at: https://freesound.org/people/cabled_mess/sounds/350983/. [Accessed 17 May 2019].

Freesound. 2019. Video Game Coin by harrietniamh. [ONLINE] Available at: https://freesound.org/people/harrietniamh/sounds/415083/. [Accessed 17 May 2019].

Freesound. 2019. Woof.mp3 by Princess6537. [ONLINE] Available at: https://freesound.org/people/Princess6537/sounds/144885/. [Accessed 17 May 2019].

Freesound. 2019. Woosh Noise 1.wav by potentjello. [ONLINE] Available at: https://freesound.org/people/potentjello/sounds/194081/. [Accessed 17 May 2019].

Textures

Unity Asset Store. 2019. 15 Original Bricks Textures - Asset Store. [ONLINE] Available at: https://assetstore.unity.com/packages/2d/textures-materials/brick/15-original-bricks-textures-72427. [Accessed 17 May 2019].

Unity Asset Store. 2019. Fantasy landscape - Asset Store. [ONLINE] Available at: https://assetstore.unity.com/packages/3d/environments/fantasy-landscape-103573. [Accessed 17 May 2019].

Unity Asset Store. 2019. Five Seamless Tileable Ground Textures - Asset Store. [ONLINE] Available at: https://assetstore.unity.com/packages/2d/textures-materials/floors/five-seamless-tileable-ground-textures-57060. [Accessed 17 May 2019].

Unity Asset Store. 2019. Grass Road Race - Asset Store. [ONLINE] Available at: https://assetstore.unity.com/packages/3d/environments/roadways/grass-road-race-46974. [Accessed 17 May 2019].

Other

GitHub. 2019. GitHub - Unity-Technologies/NavMeshComponents: High Level API Components for Runtime NavMesh Building. [ONLINE] Available at: https://github.com/Unity-Technologies/NavMeshComponents. [Accessed 17 May 2019].

Learn Everything Fast. YouTube. 2017. Rotate Camera with Mouse in Unity 3D - YouTube. [ONLINE] Available at: https://www.youtube.com/watch?v=lYIRm4QEqro. [Accessed 17 May 2019].

Appendix A

Walkies Code

Note that the following scripts have the automatically-generated Unity package code omitted from them:

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

A.1 AIBehaviour.cs

```
using UnityEngine.AI;

public class AIBehaviour : MonoBehaviour
{
    /*
     The AIBehaviour script is attached to all of the human
        AIs in the hub scene, and is responsible for all of
        their behaviour; idling, walking, running, and
        flocking.
    */

    GameObject[] humans;
    Animator move;
    int collisionAmount = 0;
    NavMeshAgent nav;
    float action;
    float speed;
    float timer, timer1, rand;
    int respawnCount;
    Vector3 destination; //destination variable holds the AI'
       s current destination

    // Start is called before the first frame update
    void Start()
    {
        humans = GameObject.FindGameObjectsWithTag("human");
           //finds all of the human AI currently in the scene
        move = gameObject.GetComponentInChildren<Animator>();
        nav = GetComponent<NavMeshAgent>(); //find the
           NavMeshAgent component to refer to
        respawnCount = 0; //AI starts off with no respawns,
           variable is used to ensure they can only respawn
           once
    }

    // Update is called once per frame
    void Update()
    {
        speed = nav.velocity.magnitude; //speed variable gets
            the AI's current speed

        if (gameObject.transform.position.y > 0.8f &&
           respawnCount == 0) //if the AI has spawned on top
           of something (i.e. where it shouldn't be),
           teleport it to new coordinates by calling the
           respawn function
        {
            respawn();
        }

        if (speed > 0.0f && speed <= 1.7f) //if the speed is
           more than 0, but not too fast (less than 1.7), set
            AI animation to walk
        {
            move.SetInteger("WalkOn", 1);
        }
        else if (speed >= 1.7f) //if the speed is more than
           1.7, set AI animation to run
        {
            move.SetInteger("WalkOn", 0);
            move.SetInteger("RunOn", 1);
        }
        else //if the speed is none of the above (thus 0 or
           less), set AI animation to idle
        {
            move.SetInteger("WalkOn", 0);
            move.SetInteger("RunOn", 0);
        }

        if (Vector3.Distance(destination, this.transform.
           position) < 3f) //if the AI is very near its
           destination for too long (indicating it's possibly
            stuck), its path gets reset and it rolls a new
           action
        {
            timer1 += Time.deltaTime;
            if (timer1 > 15f)
            {
                nav.ResetPath();
                chooseAction();
                timer1 = 0f;{AI
            }
        }

        timer += Time.deltaTime;
        rand = Random.Range(3f, 25f);
        if (rand <= timer && speed == 0.0f) //every few
           seconds, if the AI isn't moving, roll a new action
            for the AI to perform
        {
            timer = 0f;
            chooseAction();
        }
    }

    void respawn() //respawn function moves the AI to a new
       random position within the hub
    {
        respawnCount += 1;
        float spawnX = Random.Range(-28f, -10f);
        float spawnZ = Random.Range(-6f, 72f);
        this.transform.position = new Vector3(spawnX, 0.3f,
           spawnZ);
    }

    void chooseAction() //chooseAction function calls one of
       the 4 action functions; weighted differently so not
       completely random.
    {
        action = Random.Range(0f, 100f); //generates random
           number to call corresponding action
        if (action >= 0f && action <= 60f) //60% chance of
           idling
        {
            idle();
        }
        else if (action >= 60f && action <= 75f) //15% chance
            of walking
        {
            walk();
        }
        else if (action >= 75f && action <= 80f) //5% chance
           of running
        {
            run();
        }
        else if (action >= 80f) //20% chance of flocking
        {
            flock();
        }
        else
        {
            idle();
        }
    }

    void idle() //empty function to do nothing, idle the AI
    {
    }

    void walk() //walk function chooses a random coordination
        within the hub, sets the AI destination to said
       coordinate - AI walks
    {
        nav.speed = 1; //sets AI speed to walking speed
        Vector3 walkDestination;
        float destX = Random.Range(-28f, -10f);
        float destZ = Random.Range(-6f, 72f);
        walkDestination = new Vector3(destX, 0.3f, destZ);
        destination = walkDestination;
        nav.SetDestination(walkDestination); //SetDestination
            function is a built in Unity function that uses
           NavMesh pathfinding to move the AI agent to the
           specified destination
    }

    void run() //run function chooses a random coordination
       within the hub, sets the AI destination to said
       coordinate - AI runs
    {
        nav.speed = 2.5f; //sets AI speed to running speed
        Vector3 runDestination;
        float destX = Random.Range(-28f, -10f);
        float destZ = Random.Range(-6f, 72f);
        runDestination = new Vector3(destX, 0.3f, destZ);
        destination = runDestination;
        nav.SetDestination(runDestination);
    }

    void flock() //flock function directs the AI to the
       nearest human, encouraging flocking behaviour
    {
        nav.speed = 1f; //sets AI speed to walking speed
        float distance;
        float[] distances;
        float[] minMax;
        float lowest;
        int higher = 2;
        GameObject target;
        Vector3 flockDestination = this.transform.position;
           //initialisation value
        distances = new float[humans.Length];
```

```
minMax = new float[humans.Length];

for (int i = 0; i < humans.Length; i++) //for loop to
    make array of distances from current human to all
    other humans
{
    distance = Vector3.Distance(humans[i].transform.
       position, this.transform.position);
    distances[i] = distance;
}

minMax = distances; //duplicate array for storage of
   sorted values

for (int k = 0; k < minMax.Length-1; k++) //bubble
   sort through distances array, sorting from lowest
   to highest and storing in separate array (minMax)
{
    for (int l = 0; l < minMax.Length-1; l++)
    {
        if (minMax[l] > minMax[l+1])
        {
            float temp = minMax[l+1];
            minMax[l+1] = minMax[l];
            minMax[l] = temp;
        }
    }
}

lowest = minMax[1]; //array includes the person this
   script is applied to; thus the nearest person is
   the second value in the array (minMax[0] being 0)

while (lowest < 5f && higher > 10) //if the nearest
   person is too near (within 10f distance), carry on
    increasing chosen target until they're not within
    10f
{
    lowest = minMax[higher];
    higher += 1;
}

if (lowest < 5f) //if there are no people further
   than 10f away, break out of function (and thus don
   't flock)
{
    return;
}
```

```
        for (int i = 0; i < distances.Length; i++) //for loop
            finds the human the lowest distance value
           corresponds to (as determined above); sets
           destination to their position
        {
           if (lowest == distances[i])
           {
                target = humans[i];
                flockDestination = humans[i].transform.
                   position;
           }
        }

        destination = flockDestination;
        nav.SetDestination(flockDestination); //sets human's
           destination to their target
    }

    //The following function is commented out as it was used
       when I was testing around with the idea of the AI
       doing something when the player collided with them,
       such as saying increasingly offended comments
       dependent on how much the player had collided with
       them.

    /*void OnCollisionStay(Collision collider)
    {
        if (collider.gameObject.name == "Player")
        {
            collisionAmount += 1;
            if (collisionAmount >= 1 && collisionAmount <=
               10)
            {
                print("Nice to meet you!");
            }
            else if (collisionAmount >= 10 && collisionAmount
                <= 20)
            {
                print("Hello!");
            }
            else if (collisionAmount >= 20 && collisionAmount
                <= 30)
            {
                print("How rude.");
            }
            else if (collisionAmount >= 30 && collisionAmount
                <= 50)
            {
                print("Stop pushing me.");
            }
            else if (collisionAmount >= 50)
```

```
                {
                    print("Go away!");
                }
            }
        }
        */
}
```

A.2 AIDogBehaviour.cs

```
using UnityEngine.AI;

public class AIDogBehaviour : MonoBehaviour
{
    /*
    The majority of this script is duplicated from
       AIBehaviour (with the exception of the OnCollisionStay
        function found at the end of the script); and thus
       the comments can be found on the AIBehaviour script
       instead of here

    The AIDogBehaviour script is attached to all of the dog
       AIs in the hub scene, and is responsible for all of
       their behaviour; idling, walking, running, flocking,
       and informing the PlayerController script when to get
       a dog accompaniment.
    */

    GameObject[] dogs;
    Animator move;
    int collisionAmount = 0;
    NavMeshAgent nav;
    float action;
    float speed;
    float timer, timer1, rand;
    GameObject player;
    int respawnCount;
    Vector3 destination;

    // Start is called before the first frame update
    void Start()
    {
        dogs = GameObject.FindGameObjectsWithTag("dog");
        move = gameObject.GetComponentInChildren<Animator>();
        nav = GetComponent<NavMeshAgent>();
        player = GameObject.Find("Player");

        respawnCount = 0;
    }

    // Update is called once per frame
```

```
void Update()
{
    speed = nav.velocity.magnitude;

    if (gameObject.transform.position.y > 0.8f &&
       respawnCount == 0)
    {
        respawn();
    }

    if (speed > 0.1f && speed <= 1.7f)
    {
        move.SetInteger("WalkOn", 1);
    }
    else if (speed >= 1.7f)
    {
        move.SetInteger("WalkOn", 0);
        move.SetInteger("RunOn", 1);
    }
    else
    {
        move.SetInteger("WalkOn", 0);
        move.SetInteger("RunOn", 0);
    }

    if (Vector3.Distance(destination, this.transform.
       position) < 3f)
    {
        timer1 += Time.deltaTime;
        if (timer1 > 15f)
        {
            nav.ResetPath();
            chooseAction();
            timer1 = 0f;
        }
    }

    timer += Time.deltaTime;
    rand = Random.Range(3f, 25f);
    if (rand <= timer && speed == 0.0f)
    {
        timer = 0f;
        chooseAction();
    }
}

void respawn()
{
    respawnCount += 1;
    float spawnX = Random.Range(-28f, -10f);
    float spawnZ = Random.Range(-6f, 72f);
```

```
        this.transform.position = new Vector3(spawnX, 0.25f,
            spawnZ);
    }

    void chooseAction()
    {
        action = Random.Range(0f, 100f);
        if (action >= 0f && action <= 50f)
        {
            idle();
        }
        else if (action >= 50f && action <= 65f)
        {
            walk();
        }
        else if (action >= 65f && action <= 70f)
        {
            run();
        }
        else if (action >= 70f) //dogs have a higher chance
            of flocking than humans (30%)
        {
            flock();
        }
        else
        {
            idle();
        }
    }

    void idle()
    {
    }

    void walk()
    {
        nav.speed = 1;
        Vector3 walkDestination;
        float destX = Random.Range(-28f, -10f);
        float destZ = Random.Range(-6f, 72f);
        walkDestination = new Vector3(destX, 0.3f, destZ);
        destination = walkDestination;
        nav.SetDestination(walkDestination);
    }

    void run()
    {
        nav.speed = 2.5f;
        Vector3 runDestination;
        float destX = Random.Range(-28f, -10f);
        float destZ = Random.Range(-6f, 72f);
        runDestination = new Vector3(destX, 0.3f, destZ);
        destination = runDestination;
        nav.SetDestination(runDestination);
    }

    void flock()
    {
        nav.speed = 1;
        float distance;
        float[] distances;
        float[] minMax;
        float lowest;
        int higher = 2;
        GameObject target;
        Vector3 flockDestination = this.transform.position;
        distances = new float[dogs.Length];
        minMax = new float[dogs.Length];

        for (int i = 0; i < dogs.Length; i++)
        {
            distance = Vector3.Distance(dogs[i].transform.
               position, this.transform.position);
            distances[i] = distance;
        }

        minMax = distances;

        for (int k = 0; k < minMax.Length - 1; k++)
        {
            for (int l = 0; l < minMax.Length - 1; l++)
            {
                if (minMax[l] > minMax[l + 1])
                {
                    float temp = minMax[l + 1];
                    minMax[l + 1] = minMax[l];
                    minMax[l] = temp;
                }
            }
        }

        lowest = minMax[1];

        while (lowest < 5f && higher > 10)
        {
            lowest = minMax[higher];
            higher += 1;
        }

        if (lowest < 5f)
        {
            return;
        }

        for (int i = 0; i < distances.Length; i++)
        {
            if (lowest == distances[i])
            {
                target = dogs[i];
                flockDestination = dogs[i].transform.position
                    ;
            }
        }

        destination = flockDestination;
        nav.SetDestination(flockDestination);
    }

    void OnCollisionStay(Collision collider) //
       OnCollisionStay is a Unity function that takes the
       information of any collision and stores it in the
       collider variable
    {
        if (collider.gameObject.name == "Player") //if the AI
             is colliding with the player, and space is held
           down, send information on the dog type to the
           PlayerController script to activate a twin dog
           game object to accompany the player (e.g. if space
             is held down by a grey dog, the player will be
           accompanied by a grey dog game object)
        {
            if (Input.GetKeyDown(KeyCode.Space) &&
               PlayerController.hasDog == false)
            {
                PlayerController.hasDog = true; //bool
                   variable prevents the player from having
                   multiple dog accompaniments
                switch (this.name)
                {
                    case "Dog(Clone)":
                        PlayerController.dogType = 1;
                        break;
                    case "Dog (black)(Clone)":
                        PlayerController.dogType = 2;
                        break;
                    case "Dog (blonde)(Clone)":
                        PlayerController.dogType = 3;
                        break;
                    case "Dog (grey)(Clone)":
                        PlayerController.dogType = 4;
                        break;
                    case "Dog (white)(Clone)":
                        PlayerController.dogType = 5;
                        break;
                    }
                }
            }
        }
}
```

A.3 AISpawn.cs

```
public class AISpawn : MonoBehaviour
{
    /*
     The AISpawn script is attached to the terrain game
        object in the hub scene, and is responsible for
        spawning all of the AI randomly across the town; both
         dogs and humans.
    */

    [SerializeField]
    GameObject male1, male2, male3, female1, female2, female3
        , dogBrown, dogBlack, dogBlonde, dogGrey, dogWhite; //
        serialized variables to hold the AI prefabs

    GameObject[] dogs;
    GameObject[] humans;

    int humanI, dogI, spawnCount;
    float spawnX, spawnZ, spawnQ;

    // Start is called before the first frame update
    void Start()
    {
        dogs = new GameObject[5]; //dogs array holds all of
            the dog AI game object prefabs
        dogs[0] = dogBrown;
        dogs[1] = dogBlack;
        dogs[2] = dogBlonde;
        dogs[3] = dogGrey;
        dogs[4] = dogWhite;

        humans = new GameObject[6]; //humans array holds all
            of the human (male & female) AI game object
            prefabs
        humans[0] = male1;
        humans[1] = male2;
        humans[2] = male3;
        humans[3] = female1;
        humans[4] = female2;
        humans[5] = female3;
        spawnCount = Random.Range(15, 25); //generates random
            number to determine how many dogs and humans to
           spawn

        for (int i = 0; i < spawnCount; i++)
        {
            spawnX = Random.Range(-28f, -10f); //generates
               random x coordinate for current AI to spawn at
            spawnZ = Random.Range(-6f, 72f); //generates
               random z coordinate for current AI to spawn at
            spawnQ = Random.Range(0f, 360f); //generates
               random number for the current AI's angle to
               spawn as
            humanI = Random.Range(0, 6); //generates random
               number to pick a random AI prefab from the
               humans array
            Instantiate(humans[humanI], new Vector3(spawnX,
               0.3f, spawnZ), Quaternion.Euler(0f, spawnQ, 0f
               )); //spawns a random AI at a random point
               within the hub with the previously determined
               variables

            spawnX = Random.Range(-28f, -10f); //following
               code is duplicated from the human spawns,
               except using the array of dog prefabs. new
               numbers get generated so the dogs don't spawn
               in the same coordinates as the humans
            spawnZ = Random.Range(-6f, 72f);
            spawnQ = Random.Range(0f, 360f);
            dogI = Random.Range(0, 5);
            Instantiate(dogs[dogI], new Vector3(spawnX, 0.25f
               , spawnZ), Quaternion.Euler(0f, spawnQ, 0f));
        }
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

A.4 BulletinUI.cs

```
public class BulletinUI : MonoBehaviour
{
    /*
    The BulletinUI script is attached to the bulletin UI in
       the hub scene. It is responsible for determining when
        to load the bulletin UI, dependent on whether the
       player is within radius of the bulletin load
       particles, and holds down space or not. It is also
       responsible for loading/unloading specific UIs within
        the bulletin UI; button controls.
   */

   public static bool isBulletinOn = false; //public static
      variable holds state of bulletin for reference

   [SerializeField]
   GameObject bulletinUI, introductionUI, controlsUI; //
      serialized variables to hold the various UI's used for
       the bulletin UI

   PlayerController viewMode;
   GameObject player;
   GameObject particles;
   float radius;
   float keyHoldTime = 0.0f;

   // Start is called before the first frame update
   void Start()
   {
       viewMode = GameObject.Find("Player").GetComponent<
          PlayerController>(); //view mode reference to only
           allow the bulletin to load if not in view mode
       player = GameObject.Find("Player");
       particles = GameObject.Find("Bulletin particles");
       radius = particles.GetComponent<ParticleSystem>().
          shape.radius; //particle radius stored for use to
          know when the bulletin should be loaded (when
          space is held down in the particles in front of
          the bulletin)
   }

   // Update is called once per frame
   void Update()
   {
       float distance = Vector3.Distance(particles.transform
          .position, player.transform.position);  //variable
           constantly updating to check distance betwen the
          bulletin load particles and the player

       if (distance <= radius) //if the player is within the
           radius of the bulletin load particle, player is
          not in view mode, game is not paused, and space is
           held down for a few seconds, load the bulletin AI
           and pause time
       {
           if (Input.GetKey(KeyCode.Space)) //counts time
              the space key is being held for
           {
               keyHoldTime += Time.deltaTime;
           }

           if (Input.GetKey(KeyCode.Space) && keyHoldTime >=
               1f && PauseMenu.pause == false && viewMode.
              viewMode == false)
           {
               isBulletinOn = true;
               keyHoldTime = 0.0f;
               bulletinUI.SetActive(true);
               Time.timeScale = 0.0f;
           }
       }
   }

   public void introductionButton() //function applied to
      introduction button; unloads the home UI and loads the
       introduction UI
   {
       introductionUI.SetActive(true);
       bulletinUI.SetActive(false);
   }

   public void controlsButton() //function applied to
      controls button; unloads the home UI and loads the
      controls UI
   {
       controlsUI.SetActive(true);
       bulletinUI.SetActive(false);
   }

   public void homeButton() //function applied to home
      button; unloads the control & introduction UI and
      loads the home UI
   {
       introductionUI.SetActive(false);
       controlsUI.SetActive(false);
       bulletinUI.SetActive(true);
   }

   public void exitButton() //function applied to exit
      button; closes the bulletin UI and un-pauses time
   {
       isBulletinOn = false;
       bulletinUI.SetActive(false);
       Time.timeScale = 1.0f;
   }
```

```
}
```

A.5 LevelLoad.cs

```
using UnityEngine.SceneManagement;

public class LevelLoad : MonoBehaviour
{
    /*
     The LevelLoad script is attached to all of the level
        indicators in the hub scene. It is responsible for
        determining when to load the level scenes; if the
        player is within radius and holds down space, as well
         as which scene setting to load.
    */

    GameObject player;
    float keyHoldTime = 0.0f;
    string levelSetting;

    [SerializeField]
    GameObject fadeIn;
    float timer;
    bool timerOn;

    [SerializeField] //serialized field to hold music that
       plays when the player is about to start a level
    GameObject startMusic;

    void Start()
    {
        player = GameObject.Find("Player");
        timerOn = false;
        timer = 0.0f;
    }

    void Update()
    {
        float particleRadius = this.GetComponent<
           ParticleSystem>().shape.radius;
        float distance = Vector3.Distance(this.transform.
           position, player.transform.position); //gets
           distance from player to the level indicator (this)

        if (distance <= particleRadius) // check player is
           within distance of the level indicator
        {
            if (Input.GetKey(KeyCode.Space)) // counts time
               the space key is being held for
            {
                keyHoldTime += Time.deltaTime;
            }

            if (Input.GetKey(KeyCode.Space) && keyHoldTime >=
                1f) // triggers time to load level scene if
               space is currently being held for more than a
               few seconds, playing the start music and
               turning on the fade in animation
            {
                timerOn = true;
                fadeIn.SetActive(true);
                startMusic.SetActive(true);
            }
        }

        if (timerOn == true) //timer is true when level is
           ready to start after previous trigger
        {
            timer += Time.deltaTime;
            if (timer >= 2.0f) //loads level scene once a few
                seconds have passed after holding down space
               within the level indicator (allowing time for
               the fade in animation to complete)
            {

                switch (this.tag) //checks tag of current
                   level indicator to load corresponding
                   level
                {
                    case "suburb":
                        levelSetting = "SuburbLevels";
                        break;
                    case "park":
                        levelSetting = "ParkLevels";
                        break;
                    case "town":
                        levelSetting = "TownLevels";
                        break;
                    case "manhole":
                        levelSetting = "ManholeLevels";
                        break;
                }

                keyHoldTime = 0.0f;
                LevelOverMenu.isLevelOver = false; //ensures
                   game can be paused when starting a new
                   level
                SceneManager.LoadScene(levelSetting,
                   LoadSceneMode.Single);
            }
        }
    }
```

```
}
```

A.6 LevelOverMenu.cs

```
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class LevelOverMenu : MonoBehaviour
{
    /*
     The LevelOverMenu script is attached to the level over
        menu UI within the levels. It is responsible for
        deciding what to do when the level is over,
        determining how many stars (if any) the player
        receives, and deciding what to do when the return to
        hub button is pressed.
    */

    [SerializeField]
    GameObject fadeIn;

    float timer;
    bool timerOn;
    int score;
    public static bool isLevelOver = false; //public static
       bool holds reference for LevelPauseMenu to know if
       game can be paused or not (only paused if level isn't
       over)

    [SerializeField] //serialized star fields hold the star
       game objects - used to show/hide the stars dependent
       on the player's score upon level over
    GameObject star1;
    [SerializeField]
    GameObject star2;
    [SerializeField]
    GameObject star3;

    [SerializeField] //serialized footsteps field to hold
       footsteps audio
    GameObject footsteps;
    [SerializeField] //serialized level over field holds the
       level over UI
    GameObject levelOver;

    // Start is called before the first frame update
    void Start()
    {
        isLevelOver = true;
        timerOn = false; //ensures timer is reset
        timer = 0.0f;
        footsteps.SetActive(false); //turns off footsteps
           audio as soon as the level ends
        levelOver.SetActive(true); //activates the level over
            UI as soon as the level ends
    }

    // Update is called once per frame
    void Update()
    {
        score = LevelPlayerController.distanceInt; //gets the
            final score from the LevelPlayerController script
        if (score >= 200)
        {
            star1.SetActive(true); //if score is 200 or over,
                player gets 1 star
                if (score >= 500)
                {
                    star2.SetActive(true); //if score is 500
                       or over, player gets 2 stars
                        if (score >= 1000)
                        {
                            star3.SetActive(true); //if score
                                is 1000 or over, player gets
                              3 stars
                        }
                }
        }

        if (timerOn == true) //timer is true once return to
           hub button is clicked
        {
            timer += Time.deltaTime;
            if (timer >= 2.0f) //loads hub scene once a few
               seconds have passed after pressing the start
               button (allowing time for the fade in
               animation to complete)
            {
                SceneManager.LoadScene("Hub", LoadSceneMode.
                   Single);
            }
        }
    }

    public void returnToHubButton() //function applied to the
        return to hub button - turns on the fade in animation
       , and sets the timer on to trigger load of hub scene
    {
        fadeIn.SetActive(true);
        timerOn = true;
    }

}
```

A.7 LevelOverUIText.cs

```
using UnityEngine.UI;

public class LevelOverUIText : MonoBehaviour
{
    /*
     The LevelOverUIText script is responsible for retrieving
         the final score of the level, to convert and display
         in the level over UI.
    */

    LevelPlayerController player;
    public Text scoreText;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        scoreText.text = LevelPlayerController.distanceInt.
           ToString() + "m"; //gets final score from the
           LevelPlayerController script, adds 'm' for meters
    }
}
```

A.8 LevelPauseMenu.cs

```
using UnityEngine.SceneManagement;

public class LevelPauseMenu : MonoBehaviour
{
    /*
     The LevelPauseMenu script is attached to the pause menu
        UI within the levels. It is responsible for
        determining when to pause the level, as well as
        determining the action to take when either the resume
         or return to hub buttons are pressed.
    */

    public static bool pause; //public static variable holds
       pause state of level for reference
```

```
[SerializeField] //serialized field to hold the pause UI
   game object for the levels
GameObject levelPauseUI;

[SerializeField]
GameObject fadeIn;
float timer;
bool timerOn;

[SerializeField] //serialized field to hold footsteps
   audio
GameObject footsteps;

// Start is called before the first frame update
void Start()
{
    timer = 0.0f; //ensures timer is reset
    timerOn = false;
    pause = false; //ensures the game is not paused at
       runtime
}

// Update is called once per frame
void Update()
{
    if ((Input.GetKeyDown(KeyCode.P) || Input.GetKeyDown(
       KeyCode.Escape)) && LevelOverMenu.isLevelOver ==
       false) //if the escape key or p is pressed, and
       the game is not currently paused or level over,
       pause state is set to true; turning off footstep
       audio, freezing game time, and activating the
       pauseUI.
    {
        if (pause == false)
        {
            pause = true;
            footsteps.SetActive(false);
            levelPauseUI.SetActive(true);
            Time.timeScale = 0.0f;
        }
        else //if the game is currently paused, the
           previous changes are reversed by running the
           resume function (thus game is unpaused)
        {
            resumeButton();
        }
    }
```

```
        if (timerOn == true) //timer is true when return to
           hub button is pressed, loads hub scene after a few
            seconds have passed (enough time for fade in
           animation to play)
        {
            timer += Time.deltaTime;
            if (timer >= 5.0f)
            {
                SceneManager.LoadScene("Hub", LoadSceneMode.
                    Single);
            }
        }
    }

    public void resumeButton() //function applied to resume
       button, is also used if the escape or p key is pressed
        while game is currently pause. Removes pause UI,
       turns back on footstep audio, and un-freezes game time
    {
        pause = false;
        footsteps.SetActive(true);
        levelPauseUI.SetActive(false);
        Time.timeScale = 1.0f;
    }

    public void returnToHubButton() //function applied to the
        return to hub button, triggers the timer on to load
       the hub scene after a few seconds. Unfreezes time for
       next scene
    {
        timerOn = true;
        fadeIn.SetActive(true);
        Time.timeScale = 1.0f;
    }
}
```

A.9 LevelPlayerController.cs

```
using UnityEngine.SceneManagement;

public class LevelPlayerController : MonoBehaviour
{
    /*
     The LevelPlayerController script is attached to the
        player game object within all of the levels. It is
        responsible for moving the player in the levels,
        triggering the level end, playing collision audio,
        and updating the distance score.
    */

    float moveSpeed;
```

```
public static int lives; //public static allows the
   distance UI to access the variable for live visible
   updating
public static int difficulty; //public static allows
   variable to be updated upon accessing a level in
   different scene (hub)
float distance;
public static int distanceInt; //public static allows the
    distance UI to access the variable for live visible
   updating
public static bool audioPlayObstacle = false;
public static bool audioPlayPowerUp = false;
AudioSource obstacleAudio, powerUpAudio;

[SerializeField]
GameObject levelOverUI; //var to hold the UI for the
   level over
bool pause;

// Start is called before the first frame update
void Start()
{
    moveSpeed = 10.0f;
    lives = 3; //starts the player off with 3 lives
    distance = 0.0f; //starts the player off with a score
        of 0
    difficulty = PlayerController.difficulty; //gets the
       level difficulty from the hub player object

    obstacleAudio = GameObject.Find("ObstacleAudio").
       GetComponent<AudioSource>(); //sources audio
    powerUpAudio = GameObject.Find("PowerUpAudio").
       GetComponent<AudioSource>();

    switch (difficulty) //changes player speed dependent
       on level difficulty (higher levels = slower player
       )
    {
        case 1:
            moveSpeed = 3.0f;
            break;
        case 2:
            moveSpeed = 2.5f;
            break;
        case 3:
            moveSpeed = 1.5f;
            break;
        case 4:
            moveSpeed = 1.3f;
            break;
    }
}

// Update is called once per frame
void Update()
{
    pause = LevelPauseMenu.pause; //gets the pause status
        from the LevelPauseMenu script

    //LEVEL SCORE
    if (pause == false && lives > 0) //if statement
       updates distance score (if game isn't paused and
       player has lives)
    {
        distance += 0.1f;
    }
    distanceInt = Mathf.RoundToInt(distance); //makes
       distance readable; integer form

    //PLAYER MOVEMENT
    if (Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(
       KeyCode.A)) //move player left
    {
        transform.Translate(Vector3.left * Time.deltaTime
            * moveSpeed);
    }
    else if (Input.GetKey(KeyCode.RightArrow) || Input.
       GetKey(KeyCode.D)) //move player right
    {
        transform.Translate(Vector3.right * Time.
           deltaTime * moveSpeed);
    }

    //COLLISION AUDIO
    if (audioPlayObstacle == true) //trigger to play
       obstacle audio - bool variable is changed to true
       in SpawnBehaviour script
    {
        obstacleAudio.Play();
        audioPlayObstacle = false; //resets variable to
           allow for replay
    }
    else if (audioPlayPowerUp == true) //trigger to play
       powerup audio - bool variable is changed to true
       in SpawnBehaviour script
    {
        powerUpAudio.Play();
        audioPlayPowerUp = false; //resets variable to
           allow for replay
    }
    //LEVEL OVER
    if (lives <= 0) //if player has 0 lives (or less),
       level ends by bringing up the level over UI
    {
        levelOverUI.SetActive(true);
    }
}


}
```

A.10 LevelUIText.cs

```
using UnityEngine.UI;

public class LevelUIText : MonoBehaviour
{
    /*
     The LevelUIText script is attached to the level overlay
        UI in the levels, and is used to convert the life and
          distance score values from variables to strings for
        visible use in the UI.
    */

    LevelPlayerController player;
    public Text livesText;
    public Text distanceText;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        livesText.text = LevelPlayerController.lives.ToString
           ();
        distanceText.text = LevelPlayerController.distanceInt
           .ToString() + "m";
    }
}
```

A.11 MoveFloor.cs

```
public class MoveFloor : MonoBehaviour {

    /*
```

```
The MoveFloor script is attached to all of the road game
   objects found in the level scenes. This script is
   responsible for moving the roads in a conveyor-like
   fashion, and randomly generating the obstacles and
   power-ups.
*/

int difficulty;
GameObject player;
float speed;
GameObject road;
float timer = 0f;
GameObject spawn;
Vector3 spawnArea;
int spawnX;
float spawnZ;
bool pause;
int lives;
float randLower;
float randHigher;

[SerializeField]   //serialized fields to hold the
   obstacle and power up game object prefabs
GameObject hydrant, poop, manhole, treat, drink;

    // Use this for initialization
    void Start () {
    player = GameObject.Find("LevelPlayer");
    difficulty = 1; //variable initialisation
    difficulty = PlayerController.difficulty; //gets
       current level difficulty from PlayerController
       script
    lives = 3;

    switch (this.name) //switch statement decides which
       road the current road needs to spawn to the back
       of when it teleports back; like a conveyor chain.
       road 1 -> road 3 -> road 2 -> road 1.
    {
        case "Road1":
            GameObject road3 = GameObject.Find("Road3");
            road = road3;
            break;
        case "Road2":
            GameObject road1 = GameObject.Find("Road1");
            road = road1;
            break;
        case "Road3":
            GameObject road2 = GameObject.Find("Road2");
            road = road2;
            break;
    }

    switch (difficulty) //changes road speed, and rate of
        spawns dependent on level difficulty; higher the
       difficulty the higher the road speed and more
       often the spawns
    {
        case 1:
            speed = 0.3f;
            randLower = 2f;
            randHigher = 3.5f;
            break;
        case 2:
            speed = 0.4f;
            randLower = 1.5f;
            randHigher = 3f;
            break;
        case 3:
            speed = 0.5f;
            randLower = 1f;
            randHigher = 2.5f;
            break;
        case 4:
            speed = 0.6f;
            randLower = 1f;
            randHigher = 2f;
            break;
    }
}

    // Update is called once per frame
    void Update () {
    lives = LevelPlayerController.lives; //gets updated
       lives from LevelPlayerController script
    pause = LevelPauseMenu.pause; //gets pause state of
       level from LevelPauseMenu script
    Vector3 pos = transform.position; //variable pos
       holds current position of road
    float roadSize = this.GetComponent<Renderer>().bounds
       .size.z;

    //ROAD MOVEMENT
    if (pause == false && lives > 0) //road will continue
        to move as long as the game isn't paused or the
       player is out of lives
    {
        this.transform.Translate(0, 0, -speed);
    }
    if (this.transform.position.z <= (player.transform.
       position.z - roadSize/2 - 5)) //checks road
       position in the world, if completely past the
       player the road will teleport to the back of the
       last road in the 'conveyor'. the road variable
       refers to the road ahead of this road in the
       conveyor. Z position refers to the center of the
       road object, and so this checks for the z position
        minus half of the road length in comparison to
       the player position.
    {
        pos.z = road.transform.position.z + roadSize;
        transform.position = pos;
    }

    //OBJECT SPAWNING
    timer += Time.deltaTime;
    float rand = Random.Range(randLower, randHigher); //
       generates random value based on level difficulty
       to determine time range between new spawns
    if (rand <= timer && pause == false && this.transform
       .position.z >= 50) //if game isn't paused, and
       position is sufficiently far from the player, game
        will spawn a new object every x seconds
    {
        timer = 0f;
        spawnGen();
    }

    }

void spawnGen() //spawnGen function randomly spawns an
   obstacle or power-up.
{
    float spawnGen = Random.Range(0f, 100f); //generates
       random value to choose a spawn object

    if (spawnGen >= 0f && spawnGen <= 26f) //25% chance
       for spawn to be a fire hydrant
    {
        spawn = hydrant;
    }
    else if (spawnGen >= 26f && spawnGen <= 51f) //25%
       chance for spawn to be a poop
    {
        spawn = poop;
    }
    else if (spawnGen >= 51f && spawnGen <= 81f) //30%
       chance for spawn to be a manhole
    {
        spawn = manhole;
    }
    else if (spawnGen >= 81f && spawnGen <= 91f) //10%
       chance for spawn to be a dog bone
    {
        spawn = treat;
    }
    else if (spawnGen >= 91f && spawnGen <= 101f) //10%
       chance for spawn to be an energy drink
    {
        spawn = drink;
    }
    else //Spawns fire hydrant and debug text if somehow
       none of the previous spawn
    {
        spawn = hydrant;
        print(spawnGen);
        print("You shouldn't be here!");
    }

    spawnX = Random.Range(16, 25); //generates random x
       coordinate within the road
    float zLower = this.transform.position.z - 25;
    float zUpper = this.transform.position.z + 25;
    spawnZ = Random.Range(zLower, zUpper); //generates
       random z coordinate within the current road

    var obstacle = Instantiate(spawn, new Vector3(spawnX,
        0.5f, spawnZ), Quaternion.identity); //
       instantiates new spawn based on the previously
       determined variables
    obstacle.transform.parent = this.transform; //sets
       the just-spawned object's parent to be this road;
       thus the object moves along on the road
}
}
```

A.12 PauseMenu.cs

```
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    /*
     The PauseMenu script is attached to the pause menu UI
        within the hub scene. It is responsible for
        determining whether and how to pause the game, and
        what actions to take when the resume or quit buttons
        are pressed within the pause UI.
    */

    public static bool pause;
```

```
[SerializeField] //serialized field to hold the pause UI
   game object
GameObject pauseUI;
PlayerController viewMode;

// Start is called before the first frame update
void Start()
{
    pause = false; //ensures game isn't paused on runtime
    viewMode = GameObject.Find("Player").GetComponent<
       PlayerController>();
}

// Update is called once per frame
void Update()
{
   if ((Input.GetKeyDown(KeyCode.P) || Input.GetKeyDown(
      KeyCode.Escape)) && BulletinUI.isBulletinOn ==
      false && viewMode.viewMode == false) //if escape or
       p key are pressed, and not in view mode or
      bulletin on, game is paused if not already paused;
      freezing game time and turning on the pause UI
    {
        if (pause == false)
        {
            pause = true;
            pauseUI.SetActive(true);
            Time.timeScale = 0.0f;
        }
        else //if game is paused, run resume button
           function; reverts pause changes by turning off
            the pause UI, and unfreezing game time
        {
            resumeButton();
        }
    }
}

public void resumeButton() //function applied to resume
   button, also runs if the game is paused and the player
    presses escape or p key. Reverts changes made when
   game is paused
{
    pause = false;
    pauseUI.SetActive(false);
    Time.timeScale = 1.0f;
}

public void quitButton() //function applied to quit
   button, quits the application upon click
{
        Application.Quit();
    }
}
```

A.13 PlayerController.cs

```
public class PlayerController : MonoBehaviour
{
    /*
    The PlayerController script is attached to the player
       game object. This script is responsible for moving the
        player, turning on view mode, finding and storing the
        level difficulty value, respawning the player if they
        manage to get out of bounds, and spawning the player
       a dog friend.
    */

    Animator move, dogMove;
    float rotationSpeed;
    float runSpeed;
    float walkSpeed;
    GameObject cameraMouse;
    float camX, camY, camZ; //variables to hold camera reset
       rotation values
    public bool viewMode;
    public static int difficulty;
    public static bool hasDog;
    public static int dogType;
    int dogCount;
    GameObject[] levels;
    GameObject townEasy, townMedium, townHard, parkEasy,
       parkMedium, parkHard, suburbsEasy, suburbsMedium,
       suburbsHard;
    GameObject bulletinParticles, townManhole, parkManhole,
       suburbManhole;
    GameObject introductionUI;
    float townManholeD, parkManholeD, suburbManholeD;
    float distance;
    float particleRadius;
    float smallParticleRadius;

    [SerializeField]
    GameObject footstep, woof, dog, dogBlack, dogBlonde,
       dogGrey, dogWhite;

    // Start is called before the first frame update
    void Start()
    {
        hasDog = false;
        dogType = 0;
        dogCount = 0;
```

```
move = gameObject.GetComponentInChildren<Animator>();
    //reference to player animator
dogMove = gameObject.GetComponentInChildren<Animator
    >();
cameraMouse = GameObject.Find("Camera");
rotationSpeed = 200.0f;
runSpeed = 4.0f;
walkSpeed = 1.0f;
camX = 20f;
camY = 0.0f;
camZ = 0.0f;
cameraMouse.GetComponent<MouseCamera>().enabled =
    false;
viewMode = false;
distance = 0.0f;
difficulty = 0;
introductionUI = GameObject.Find("IntroductionUI");

//LEVELS
bulletinParticles = GameObject.Find("Bulletin
    particles");
townEasy = GameObject.Find("TownEasy");
townMedium = GameObject.Find("TownMedium");
townHard = GameObject.Find("TownHard");
parkEasy = GameObject.Find("ParkEasy");
parkMedium = GameObject.Find("ParkMedium");
parkHard = GameObject.Find("ParkHard");
suburbsEasy = GameObject.Find("SuburbsEasy");
suburbsMedium = GameObject.Find("SuburbsMedium");
suburbsHard = GameObject.Find("SuburbsHard");
levels = new GameObject[9]; //levels array stores
    references to all of the level indicators within
    the hub
levels[0] = townEasy;
levels[1] = parkEasy;
levels[2] = suburbsEasy;
levels[3] = townMedium;
levels[4] = parkMedium;
levels[5] = suburbsMedium;
levels[6] = townHard;
levels[7] = parkHard;
levels[8] = suburbsHard;

townManhole = GameObject.Find("townManhole");
parkManhole = GameObject.Find("parkManhole");
suburbManhole = GameObject.Find("suburbManhole");

particleRadius = townEasy.GetComponent<ParticleSystem
    >().shape.radius;
smallParticleRadius = bulletinParticles.GetComponent<
    ParticleSystem>().shape.radius;
```

```
        if (StartMenu.firstGame == false) //prevents
            introduction text from appearing if the player is
            returning after a level
        {
            introductionUI.SetActive(false);
        }
    }

    // Update is called once per frame
    void Update()
    {
        //trigger for introduction text to fade away upon
            first load
        if ((Input.GetKey(KeyCode.UpArrow) || Input.GetKey(
            KeyCode.W) || Input.GetKey(KeyCode.DownArrow) ||
            Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.
            LeftArrow) || Input.GetKey(KeyCode.A) || Input.
            GetKey(KeyCode.RightArrow) || Input.GetKey(KeyCode
            .D) || Input.GetKey(KeyCode.V) || Input.GetKey(
            KeyCode.P) || Input.GetKey(KeyCode.Escape)) &&
            StartMenu.firstGame == true)
        {
            StartMenu.firstGame = false;
            introductionUI.GetComponent<Animator>().enabled =
                true;
        }

        //VIEW MODE (free view with mouse, disables player
            movement)
        if (Input.GetKeyDown(KeyCode.V))
        {
            if (viewMode == false && PauseMenu.pause == false
                ) //goes into view mode if not already, game
                isn't paused, and v key is pressed
            {
                viewMode = true;
                cameraMouse.GetComponent<MouseCamera>().
                    enabled = true;
            }
            else //if in view mode, and v is pressed, toggle
                view mode off & reset camera position
            {
                viewMode = false;
                cameraMouse.GetComponent<MouseCamera>().
                    enabled = false;
                camY = this.transform.eulerAngles.y; //update
                     camera's Y reset position rotation to be
                    that of the player's current Y rotation
                cameraMouse.transform.eulerAngles = new
                    Vector3(camX, camY, camZ); //reset camera
                    to be facing player
            }
        }

        //PLAYER MOVEMENT
        if ((Input.GetKey(KeyCode.UpArrow) || Input.GetKey(
            KeyCode.W)) && viewMode == false && PauseMenu.
            pause == false) //moves player forward if up arrow
            or w is pressed, game isn't paused, and not in
            view mode
        {
            move.SetInteger("RunOn", 1); //turns on run
                animation
            footstep.SetActive(true); //footstep audio is on
                when moving
            transform.Translate(Vector3.forward * runSpeed *
                Time.deltaTime); //forward control

            if (hasDog == true) //if player has a dog, sets
                dog animation to run also
            {
                dogMove.SetInteger("RunOn", 1);
            }
        }
        else //when idle
        {
            move.SetInteger("RunOn", 0); //turns off run
                animation
            footstep.SetActive(false); //turns off footstep
                audio

            if (hasDog == true) //if player has a dog, sets
                dog animation to idle also
            {
                dogMove.SetInteger("RunOn", 0);
            }
        }

        if ((Input.GetKey(KeyCode.DownArrow) || Input.GetKey(
            KeyCode.S)) && viewMode == false) //moves player
            backwards if down arrow or s is pressed, game isn'
            t paused, and not in view mode
        {
            move.SetInteger("WalkOn", 1); //turns on walk
                animation
            transform.Translate(Vector3.back * walkSpeed *
                Time.deltaTime); //backward control
            if (hasDog == true) //if player has a dog, sets
                dog animation to walk also
            {
                dogMove.SetInteger("WalkOn", 1);
            }
        }
        else
        {
            move.SetInteger("WalkOn", 0); //turns off run
                animation
            if (hasDog == true) //if player has a dog, sets
                dog animation to idle also
            {
                dogMove.SetInteger("WalkOn", 0);
            }
        }

        if ((Input.GetKey(KeyCode.LeftArrow) || Input.GetKey(
            KeyCode.A)) && viewMode == false) //rotate camera
            left if not in view mode, and left arrow or a is
            pressed
        {
            transform.Rotate(Vector3.down * rotationSpeed *
                Time.deltaTime);
        }
        else if ((Input.GetKey(KeyCode.RightArrow) || Input.
            GetKey(KeyCode.D)) && viewMode == false) //rotate
            camera right if not in view mode, and right arrow
            or d is pressed
        {
            transform.Rotate(Vector3.up * rotationSpeed *
                Time.deltaTime);
        }

        //KILL ZONE (out of bounds)
        if (transform.position.x <= -35 || transform.position
            .x >= -2 || transform.position.z <= -11 ||
            transform.position.z >= 78)
        {
            transform.position = new Vector3(-19.59f, 0.285f,
                  -4.885f); //teleports player back to start if
                  they somehow get out of bounds
        }


        //LEVEL DIFFICULTY
        if (Input.GetKeyDown(KeyCode.Space)) { //following
            code runs if space is pressed

            //SETTING NORMAL LEVEL DIFFICULTIES
```

```
for (int i = 0; i < levels.Length; i++) //for
   loops through the array of all the level
   indicators in the scene; if player is within
   radius of any of them, sets difficulty
   corresponding to index of array (first 3
   indexes are easy level indicators, next 3
   medium level indicators, last 3 hard level
   indicators)
{
    distance = Vector3.Distance(levels[i].
       transform.position, this.transform.
       position);
    if (distance <= particleRadius) // check
       player is within distance of the level
       indicator
    {
        if (i >= 0 && i <= 2)
        {
            difficulty = 1; //easy difficulty
               value
        }
        else if (i >= 3 && i <= 5)
        {
            difficulty = 2; //medium difficulty
               value
        }
        else if (i >= 6 && i <= 8)
        {
            difficulty = 3; //hardest difficulty
               value
        }
    }
}

//SETTING MANHOLE DIFFICULTY
townManholeD = Vector3.Distance(townManhole.
   transform.position, this.transform.position);
   //gets distance of player to all 3 manholes in
    the hub scene
parkManholeD = Vector3.Distance(parkManhole.
   transform.position, this.transform.position);
suburbManholeD = Vector3.Distance(suburbManhole.
   transform.position, this.transform.position);
if (townManholeD <= smallParticleRadius ||
   parkManholeD <= smallParticleRadius ||
   suburbManholeD <= smallParticleRadius) //
   check player is within radius of any of the
   manholes; if so, sets difficulty to hardest
{
    difficulty = 4;
}
```

```
            //GETTING A DOG
            if (hasDog == true && dogCount == 0) //player
               only obtains a dog if they currently don't
               have one, and hasDog has been triggered on
               from the AIDogBehaviour script
            {
                woof.SetActive(true); //plays bark audio when
                    dog is obtained
                dogCount += 1; //updates dogCount variable to
                    prevent player from getting any more dogs
                switch (dogType) //dogType variable is
                   updated once trigger has been performed in
                    the AIDogBehaviour script. Corresponding
                   dog colour game object is set active as
                   per below
                {
                    case 1:
                        dog.SetActive(true);
                        dogMove = dog.GetComponentInChildren<
                           Animator>();
                        break;
                    case 2:
                        dogBlack.SetActive(true);
                        dogMove = dogBlack.
                           GetComponentInChildren<Animator>()
                           ;
                        break;
                    case 3:
                        dogBlonde.SetActive(true);
                        dogMove = dogBlonde.
                           GetComponentInChildren<Animator>()
                           ;
                        break;
                    case 4:
                        dogGrey.SetActive(true);
                        dogMove = dogGrey.
                           GetComponentInChildren<Animator>()
                           ;
                        break;
                    case 5:
                        dogWhite.SetActive(true);
                        dogMove = dogWhite.
                           GetComponentInChildren<Animator>()
                           ;
                        break;
                }
            }
        }

    }
```

```
}
```

A.14 SpawnBehaviour.cs

```
public class SpawnBehaviour : MonoBehaviour
{
    /*
     The SpawnBehaviour script is attached to all the spawned
          obstacles and power—ups within the levels. It is
        responsible for detecting collisions between the
        object and either the killbox (which destroys the
        object), or the player or dog (which determines
        whether a life should be lost or added etc.).
     */

    int lives;

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }

    void OnTriggerEnter(Collider collision) //OnTriggerEnter
       is a Unity function that holds information about
       collisions in the collision variable
    {
        if (collision.gameObject.name == "KillBox") //
           destroys game object if it collides with the
           killbox; ie. goes past the player (prevents
           objects from making a second round on the conveyor
           )
        {
            Destroy(gameObject);
        }

        if (collision.gameObject.name == "LevelPlayer" ||
           collision.gameObject.name == "Dog" &&
           LevelPauseMenu.pause == false) //Triggers
           following code if game object colliders with the
           player or the dog, and the game isn't paused
        {
            lives = LevelPlayerController.lives; //gets
               current lives
```

```
switch (this.name) //gets the name of the current
    game object to refer to. If object is a fire
   hydrant, manhole, or poop, and the player has
   1 or more lives, player loses a life and
   obstacle audio plays. If object is an energy
   drink or dog bone, and the player has less
   than 5 lives, player gains a life and power up
    audio plays.
{
    case "firehydrant Variant(Clone)":
        if (lives >= 1)
        {
            lives -= 1;
            LevelPlayerController.
               audioPlayObstacle = true;
        }
        break;
    case "manhole Variant(Clone)":
        if (lives >= 1)
        {
            lives -= 1;
            LevelPlayerController.
               audioPlayObstacle = true;
        }
        break;
    case "poop Variant(Clone)":
        if (lives >= 1)
        {
            lives -= 1;
            LevelPlayerController.
               audioPlayObstacle = true;
        }
        break;
    case "energydrink Variant(Clone)":
        if (lives < 5)
        {
            lives +=1;
            LevelPlayerController.
               audioPlayPowerUp = true;
        }
        break;
    case "bone Variant(Clone)":
        if (lives < 5)
        {
            lives += 1;
            LevelPlayerController.
               audioPlayPowerUp = true;
        }
        break;
}
```

```
            LevelPlayerController.lives = lives; //updates
               lives
            Destroy(gameObject); //game object is destroyed
               upon collision with the player or dog.
               Prevents from accidentally double-colliding
               with the same object
        }
    }
}
```

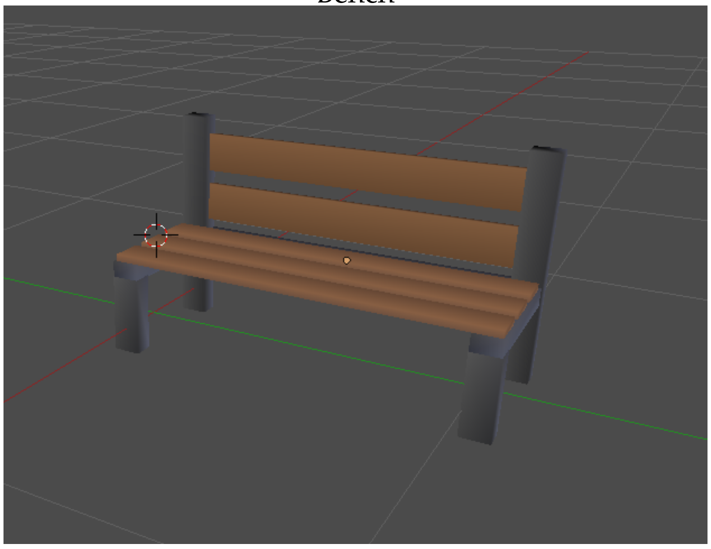
A.15 StartMenu.cs

```
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    /*
     The StartMenu script is attached to the start menu in
        the start scene. It is responsible for determining
        what to to when the buttons within the start UI are
        clicked.
    */

    [SerializeField]
    GameObject fadeIn; //allows for reference to fade in
       animation object

    float timer;
    bool timerOn;
    public static bool firstGame = true; //public static to
       hold game state of whether the player has loaded the
       hub before or not (and thus to inform the scene
       whether the introduction text should show or not)

    // Start is called before the first frame update
    void Start()
    {
        timerOn = false; //ensures timer is reset
        timer = 0.0f;
    }

    // Update is called once per frame
    void Update()
    {
        if (timerOn == true) //timer starts when true (thus
           when start button is pressed)
        {
            timer += Time.deltaTime;
            if (timer >= 2.0f) //loads hub scene once a few
               seconds have passed after pressing the start
               button (allowing time for the fade in
               animation to complete)
            {
                SceneManager.LoadScene("Hub", LoadSceneMode.
                    Single);
            }
        }

    }

    public void startButton() //function that is applied to
       the start button
    {
        fadeIn.SetActive(true); //fadeIn animation activates
           once start button is pressed
        timerOn = true;   //starts timer, to allow time for
           fade animation to run rather than immediately
           loading the hub scene
    }

    public void quitButton() //function that is applied to
       the quit button; quits application upon click
    {
        Application.Quit();
    }
}
```

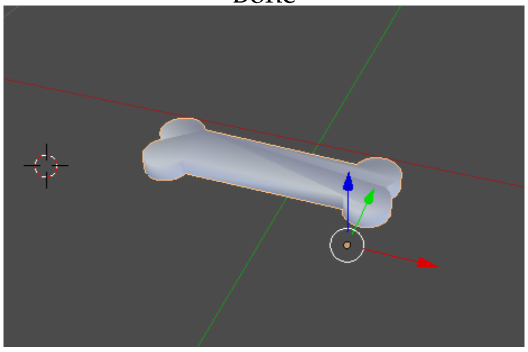
Appendix B

Walkies Assets

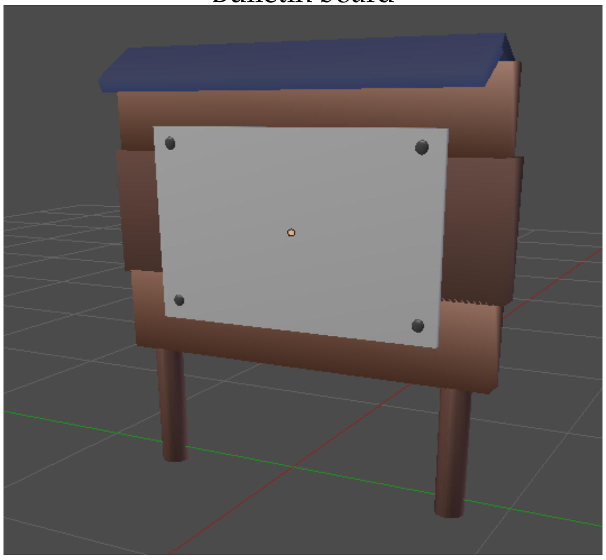
Bench



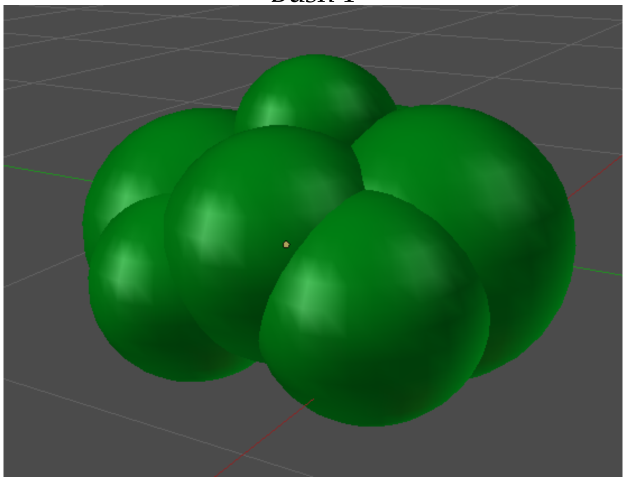
Bone



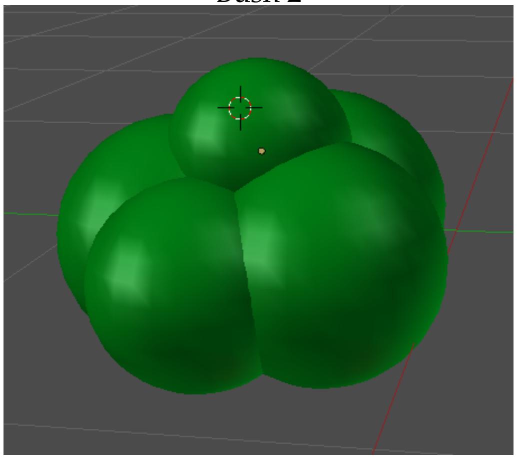
Bulletin board



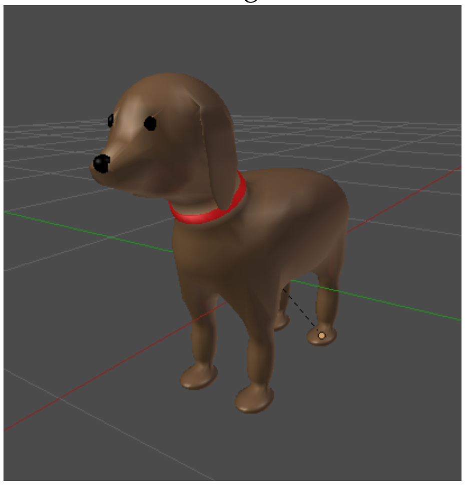
Bush 1



Bush 2

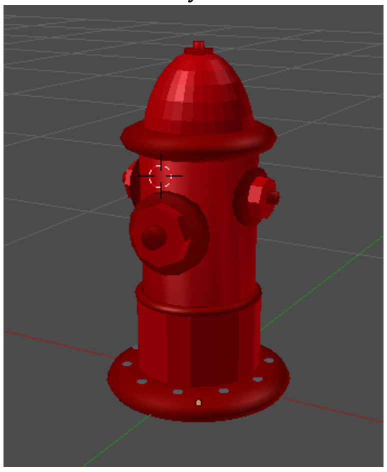


Dog



Energy drink

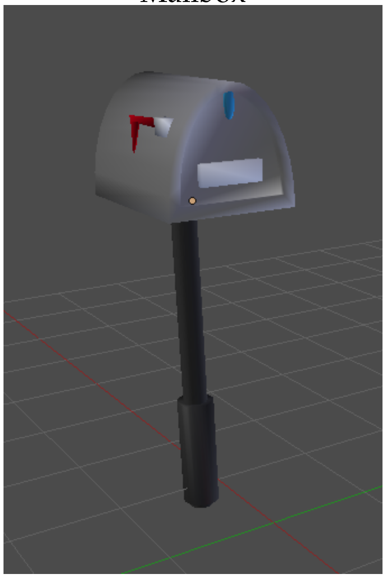
Fire hydrant



Flowers

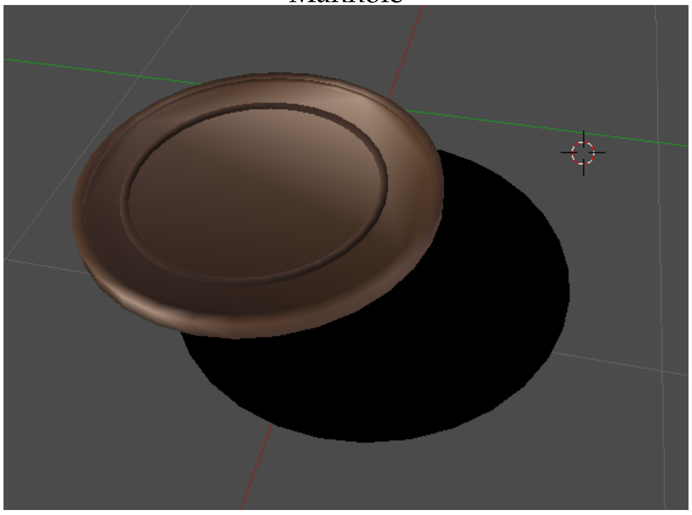
Lamppost

Mailbox

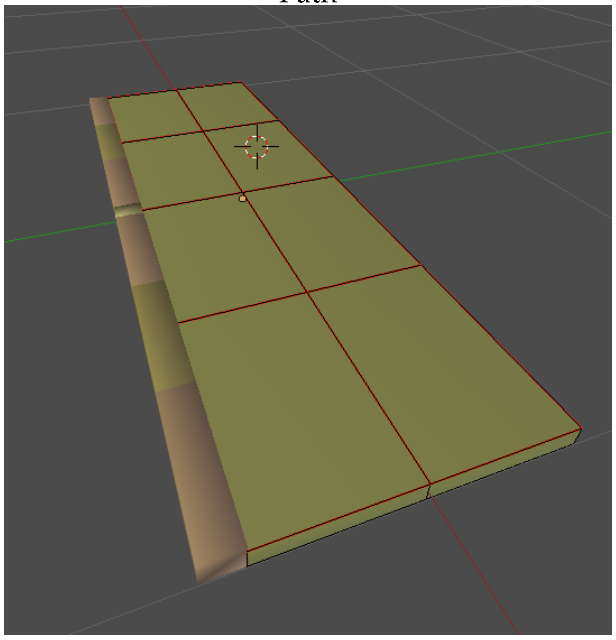


Man

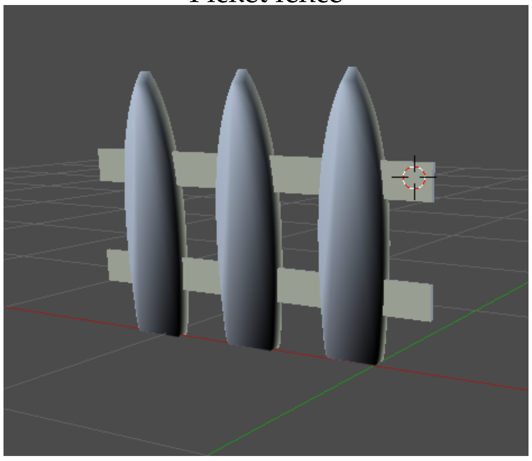
Manhole



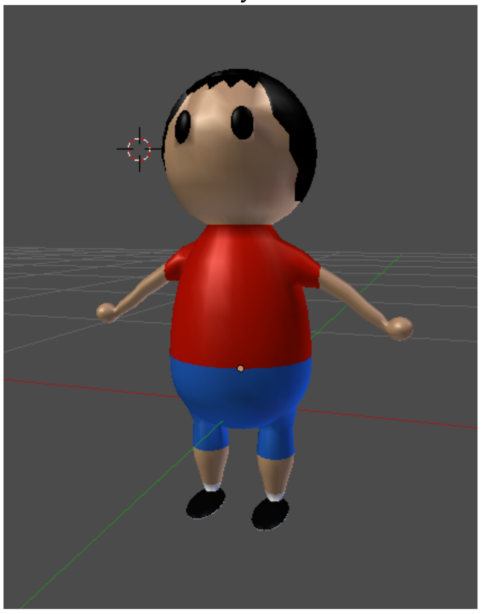
Path



Picket fence

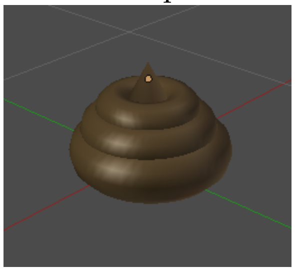


Player

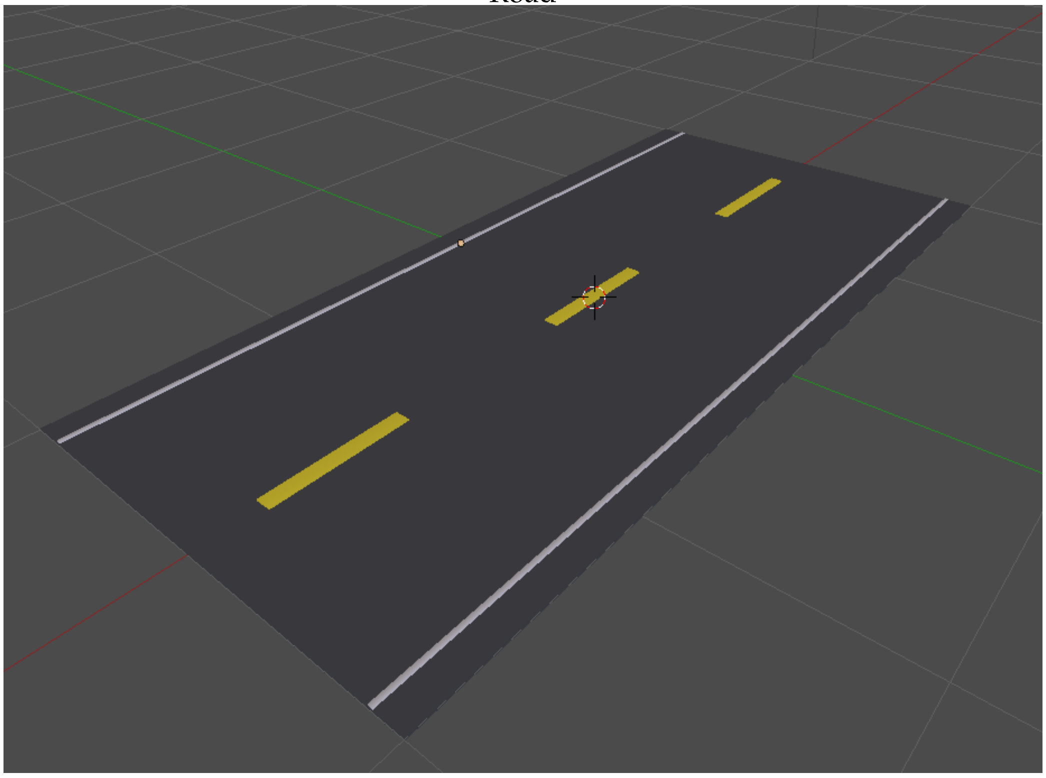


Player - baseball cap

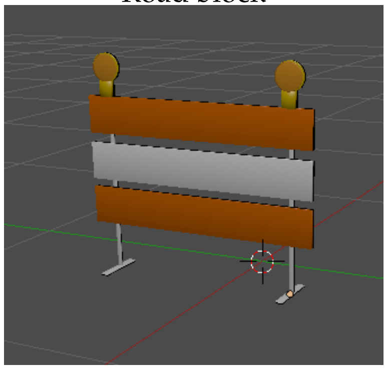
Poop



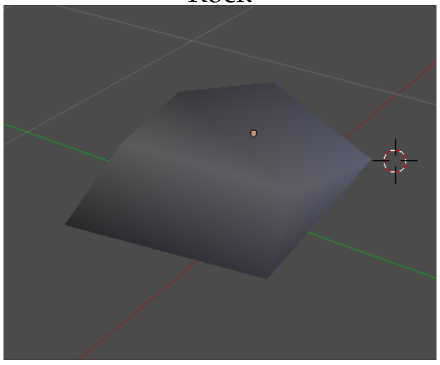
Road



Road block



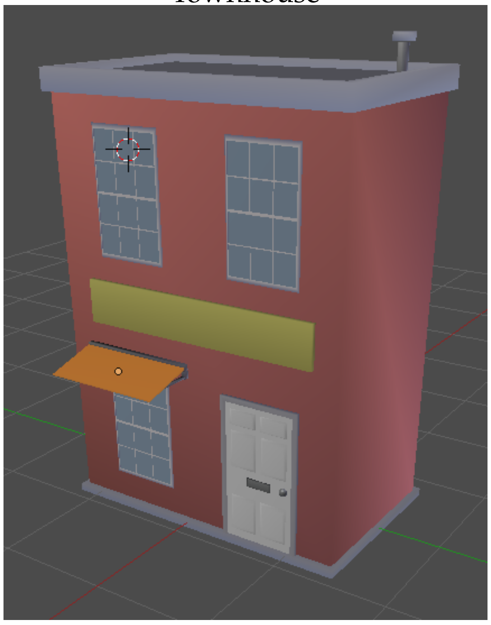
Rock



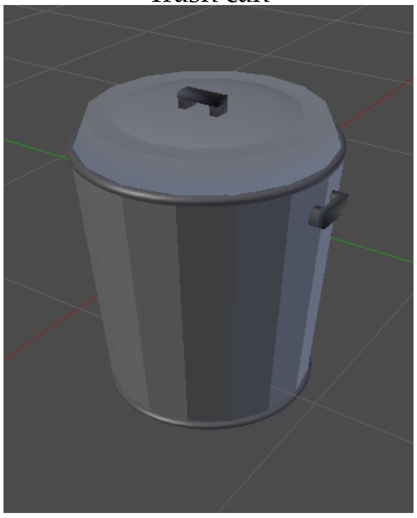
Suburban house

Townhall

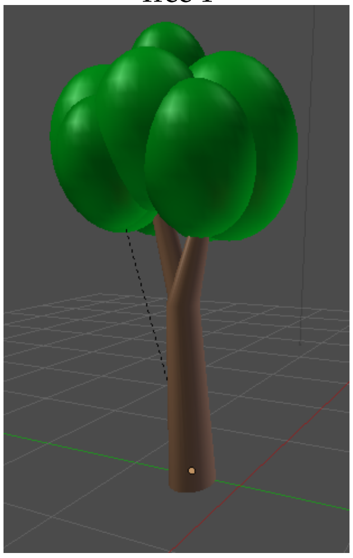
Townhouse



Trash can

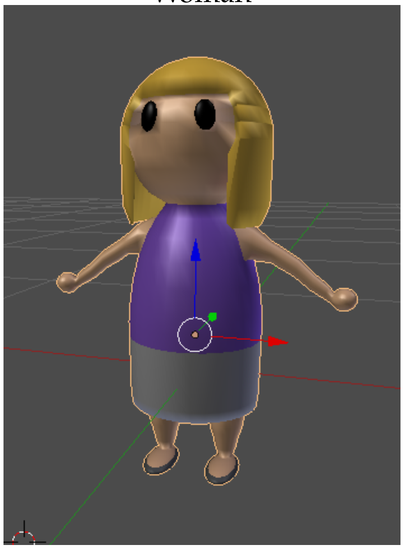


Tree 1



Tree 2

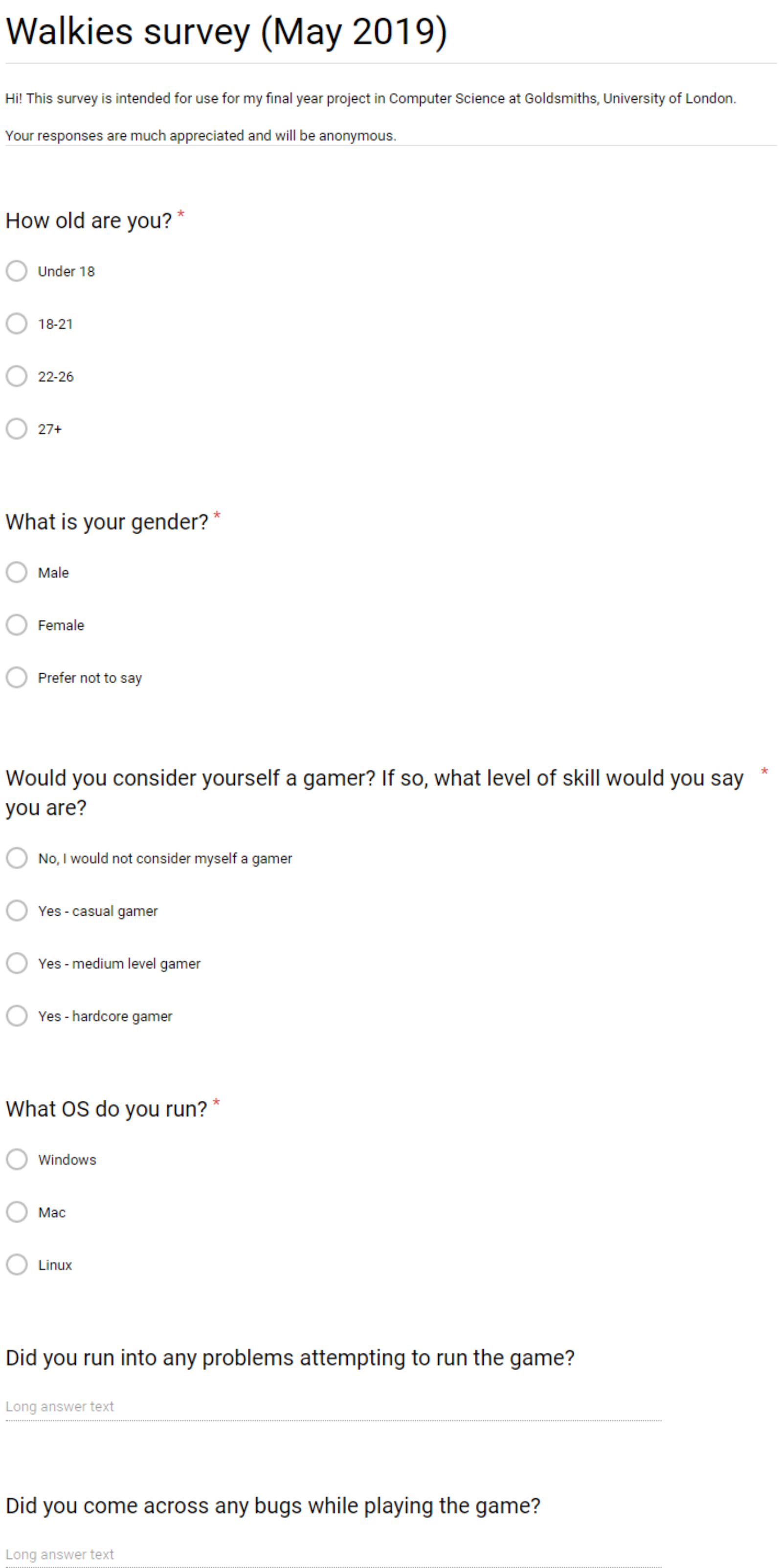
Woman



Wood fence

Appendix C

Walkies Survey



Walkies survey (May 2019)

Hi! This survey is intended for use for my final year project in Computer Science at Goldsmiths, University of London.

Your responses are much appreciated and will be anonymous.

How old are you? *

- ◯ Under 18
- ◯ 18-21
- ◯ 22-26
- ◯ 27+

What is your gender? *

- ◯ Male
- ◯ Female
- ◯ Prefer not to say

Would you consider yourself a gamer? If so, what level of skill would you say you are? *

- ◯ No, I would not consider myself a gamer
- ◯ Yes - casual gamer
- ◯ Yes - medium level gamer
- ◯ Yes - hardcore gamer

What OS do you run? *

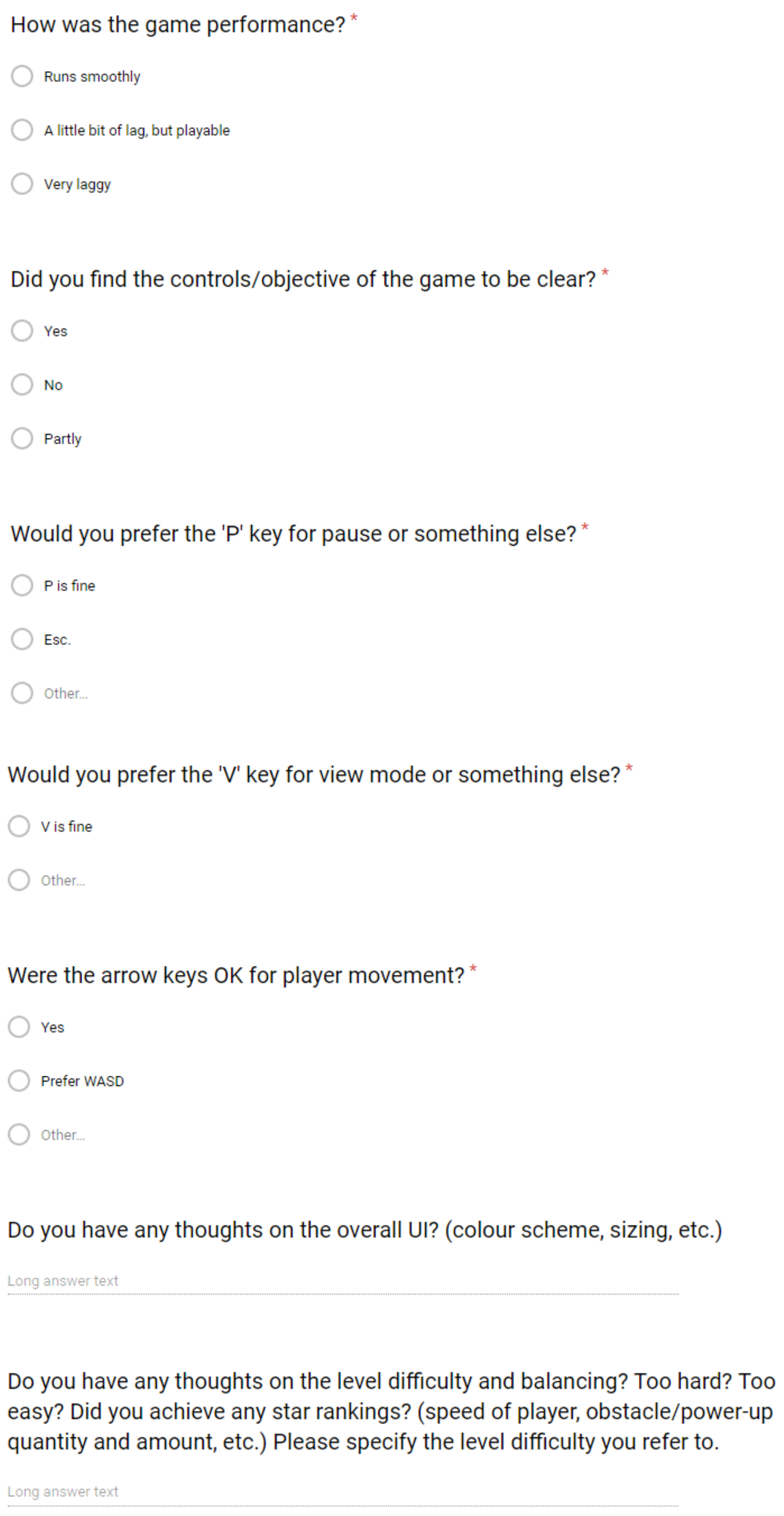
- ◯ Windows
- ◯ Mac
- ◯ Linux

Did you run into any problems attempting to run the game?

Long answer text

Did you come across any bugs while playing the game?

Long answer text



How was the game performance? *

○ Runs smoothly

○ A little bit of lag, but playable

○ Very laggy

Did you find the controls/objective of the game to be clear? *

○ Yes

○ No

○ Partly

Would you prefer the 'P' key for pause or something else? *

○ P is fine

○ Esc.

○ Other...

Would you prefer the 'V' key for view mode or something else? *

○ V is fine

○ Other...

Were the arrow keys OK for player movement? *

○ Yes

○ Prefer WASD

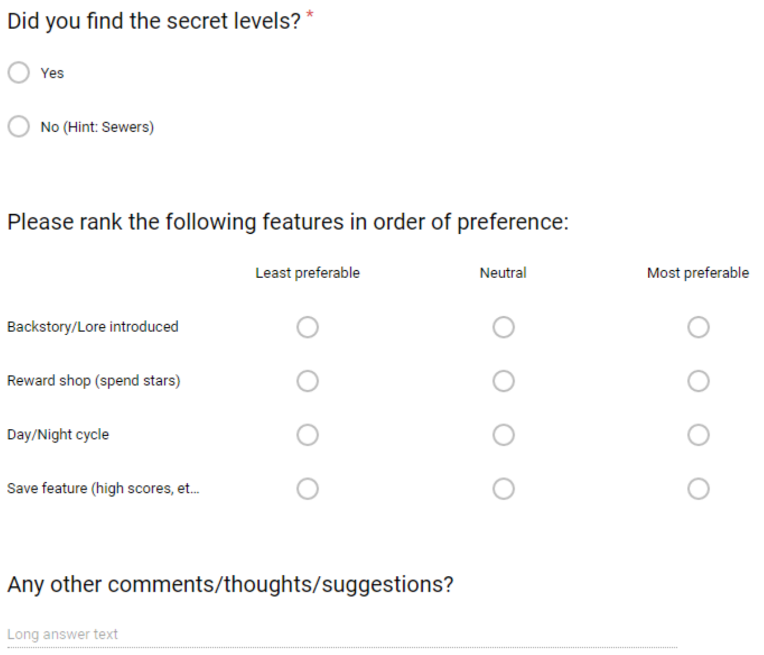
○ Other...

Do you have any thoughts on the overall UI? (colour scheme, sizing, etc.)

Long answer text

Do you have any thoughts on the level difficulty and balancing? Too hard? Too easy? Did you achieve any star rankings? (speed of player, obstacle/power-up quantity and amount, etc.) Please specify the level difficulty you refer to.

Long answer text



Did you find the secret levels? *

○ Yes

○ No (Hint: Sewers)

Please rank the following features in order of preference:

	Least preferable	Neutral	Most preferable
Backstory/Lore introduced	○	○	○
Reward shop (spend stars)	○	○	○
Day/Night cycle	○	○	○
Save feature (high scores, et...	○	○	○

Any other comments/thoughts/suggestions?

Long answer text

Appendix D

Weekly Logs

D.1 Week 1 – w/c 21/01/2019

Over the Christmas holidays, I started to look into the feasibility of me using Blender for modelling in my project. I looked at a few tutorials, and also looked into if Maya was feasible – you have to pay for it, so I decided to stick with Blender.

For the first two weeks of term, I wrote down a more solidified plan for my game:

- Scrapped having a customisable character (unless I have time down the line)
- Added a 3-star ranking system
- Changed the design to have a 3D hub world in which you walk to various levels (signified by a coloured ring in which you step into – green = easy, yellow = medium, red = hard) – 2D pixel style levels in which you walk a dog whilst avoiding obstacles (manholes, etc.) and gain powerups (dog treats, etc.)
- Potential for having NPC AI around the hub world (as I will be learning about AI in this term's module, Games AI Programming)
- Potential for having different states of the hub world (decrepit = no scores on any levels, thriving = 3 stars on all levels)

For the second week back (first week of labs), we discussed using LaTeX for our final report. Unfortunately, I was preoccupied and rather busy this week, and my supervisor is in Paris next week, so I will be meeting with him the week after – giving me plenty of time to provide substantial material to go over with him.

Next week I plan to look further into Blender – downloading it and testing it out myself, as well as playing around with C# in Unity in order to get more comfortable with the language and scripting in Unity.

D.2 Week 2 – w/c 28/01/2019

The past week I downloaded Blender and played around with it. Additionally, I viewed more in-depth character tutorials including:

- Modelling
- UV Mapping
- Texturing
- Rigging

- Animation

These gave me an idea of the timeframe it would take me to make models – I hope to make a base character model, with small adjustments (e.g colours, hair length) for different NPCs. To help me in visualising models, I listed all the models I would need to make for my game (such as NPCs, cars, dogs, trees, houses – town & suburban – fences, flowers, fire hydrants, trash cans, bones, energy drinks, manholes)

I also planned out a design document, as this weekend I will be making a detailed design document to go through with my supervisor on Monday, including gameplay features, level design, map design, other features (e.g. day/night cycle), and additional features (features that I would like to include if I have time).

In my Games AI module, we learned about Finite State Machines and Behaviour Trees, helping me in thinking about how AI would work in the game. I also looked at a car AI tutorial to see if that is feasible – likely to be an additional feature, rather than base.

Next week I plan to make a base human model in which I can then make the other NPC & player character model off of – and hopefully animate. By starting with the most complex model, this will get the hardest part out of the way and get me used to modelling lesser complexity environment models. In general, I'd like to get the modelling out of the way so I can focus on coding for the majority of the project time.

D.3 Week 3 – w/c 4/02/2019

This week, I finished writing up a detailed design document for my project and went over it in person with my supervisor. Generally, it looked good, however he suggested to pick a specific technical aspect to expand on – whether that be something like procedural generation, or AI. As I'm doing a module in Games AI alongside, I will probably choose to focus on this in order to maximise my marks. My detailed design document will likely be included somewhere in my final submission – e.g. in the appendix.

I also sketched out the character and environment designs for my models and started work on the base character model in Blender. I would like to get this finished by next week.

My supervisor said it would also be nice if I could get a basic prototype working first, to see how it works in gameplay. I agree, and thus plan on making a basic prototype for the levels next week – a basic infinite runner where you gather powerups whilst avoiding obstacles. This would also be good timing in order to talk about the prototype in my preliminary project report.

I started thinking about how the prototype will work in terms of specific logic – likely, I will have 2 (or 3) ground objects, which queue in front of the character when they reach x point (in order to create an infinite looking ground). The powerup and obstacle objects will have the possibility of spawning in 1 of x (around 5) columned points at any given time (randomised). Once spawned, they will move towards the player until they reach a certain point (behind the player), at which they de-spawn.

D.4 Week 4 – w/c 11/02/2019

Unfortunately, I did not get as much done this week as I would have liked, as I have been preoccupied and get most of the work done on weekends. I did, however, start a new Unity project in which I touched upon testing out the level logic for my game. I plan to finish that this weekend, so I have something to show my supervisor next week, and to discuss in my preliminary project report.

I also looked through the preliminary project report requirements and started planning out what I will write for each part, as well as ensuring I will have solid progress complete by the beginning of next week – which I can discuss in my PPL.

In my Games AI module this week, we learned about pathfinding – a topic which will be useful in regards to my implementation of AI in my game.

D.5 Week 5 – w/c 18/02/2019

This week I focussed on writing my preliminary project report. Although my already-made design document helped with the aims and objectives section, I still had to detail out my plans for the rest of my project, as well as delve into what I've done so far.

I aimed to finish the first level prototype this week – however this did not get finished due to me underestimating the difficulty of it, as well as my preoccupation with the preliminary project report. I will make sure to finish this prototype by the end of February at the latest, in order for me to not lag too far behind on my project.

My supervisor sent me some helpful game design document templates, and we discussed what I should include within my preliminary project report. Additionally, we discussed available software to use for LaTex – should I not want to learn the syntax myself – and other forms of Git for version control.

D.6 Week 6 – w/c 25/02/2019

I do not have much to report on this week due to being busy with other preoccupations. Throughout the week I have just been working on the basic level implementation – infinite runner style. I aim to complete this over the weekend, as detailed in my preliminary project report, and will be meeting with my supervisor next week to discuss it.

D.7 Week 7 – w/c 4/03/2019

This week I have done a few various random things – not necessarily sticking to my plan of finishing the level prototype. While I have expanded on the logic used for the level prototype, and considered other ways to go about it, I have mainly been working on other aspects of the game this week. This has included transportation between the world hub and the levels, creating funky particle systems for this transportation as indicators, and work on the world hub itself – blocking out the layout.

Additionally, I have been going through a Unity course on Udemy, in an attempt to become more comfortable with all aspects of the engine. Lastly, as we went through procedural content generation in this week's Games AI lecture, I have considered and briefly planned out a way in which I could add some procedural generation to the hub creation – such as splitting it up into prefab chunks, and generating from there in a fixed grid (spawn x town chunks, x suburb chunks, x park chunks next to each other, for x in a grid).

D.8 Week 8 – w/c 11/03/2019

This week has been another busy week for me, and thus I have not done as much as I would have hoped. Instead I have been watching various Brackeys videos, with tips on game design and Unity. Additionally, I have made a custom skybox for my project.

I met up with my supervisor this week and we just checked in that things are going ok. Although I'm a little bit behind, I'm not worried as I will have much more spare time in the upcoming weeks to really sit down and power through the work – the current issue is my lack of time, rather than not knowing what to do.

Lastly, I looked into and discussed more about character animation in the lab session - keyframes needed, and considering the detail needed for animations and character controller.

D.9 Week 9 – w/c 18/03/2019

This week I finally managed to complete my level implementation. While it was intended to be a prototype, it's closer to the final version and just needs tweaking regarding level difficulty and different settings. There is an infinite road, randomly spawned obstacles & powerups (each object having a different weighting), lives, and ability to move the character left and right. All that is left to do is to finetune numerical values for the various difficulties – affecting the speed of the level, number & rarity of obstacles & powerups.

I did not meet my supervisor this week as he in in Asia. However, I will be meeting him next week and hope to be able to look at previous examples of game-specific final year projects in order to get an idea of the level of complexity I need to implement to get a good grade.

As I am behind on my plan outlined in my preliminary project report, I have re-planned the next few weeks in the schedule, as follows:

- 25/03-31/03: Create and finalise all GUIs – including but not limited to: start/-pause menus, and bulletin board. Start modelling all assets – human and environment. Clean up & finalise level implementations (namely difficulty).
- 1/04-07/04: Finish up modelling and anything not finishing in the previous week. Layout everything in the world – asset-wise – and finish styling for the whole game. Should animating human models prove to be more difficult than I anticipate, I plan to fall back on using capsule models – these bob around instead of walking with arms and legs.
- 8/04-14/04: Start working on AI.
- 15/04-21/04: Finish working on AI.
- 22/04-28/04: Finish up the whole game, giving me an opportunity to add any small extra features, play around with procedural generation, or finish anything that I did not already in the previous weeks. If all is good, start on the report. I am also away for half of this week so will not have as much time. Ideally, I would like to get a final draft complete in good time to get feedback from my supervisor.
- 29/04-17/05: Continue & finish writing report. Hand in.

Now that I have a better idea of how long it will take me to complete various tasks, in addition to the extra time I will have over Easter, I can make a more accurate plan- although my previous plan outlined the basic deadlines that have not changed. I should note that testing is an ongoing process throughout all of development, as no specific time is set aside for it.

For next week, I will just be working on the tasks outlined in the plan above – prioritising finishing all the details in the level implementations.

D.10 Week 10 – w/c 25/03/2019

This week I finished fine-tuning the values for my level implementations, affecting the 3 difficulties. This involved constant testing and playing around with various numbers to see which felt right – replaying the level to see if scoring was consistent enough, and that the hard difficulty was a sufficient challenge.

I have plans to meet my supervisor at the start of next week, where I hope to show him my progress to date and view previous project examples.

Additionally, I started modelling assets, using both Blender for basic environment assets (not character models yet as they are more complex), and Unity's tree maker in order to ensure my tree assets can sway in the wind with Unity terrain wind zone feature.

The GUIs have been implemented in a basic sense – requiring styling to finish. I need to find an appropriate font, and to create a logo using vector graphics. I also plan on making a canvas to have an animated start screen.

Although this is the last weekly log, the next month will be the time for me to buckle down working on my project, as I am yet to start writing my report or creating the AIs for my hub world (as outlined in my revised plan to start in approximately 2 weeks).

Appendix E

Original Proposal

E.1 Introduction

I propose to make a first-person, single-player, offline video game, centered around dog walking. It will be made using Unity 3D, for PC. The game will have a brighter, cartoonish/stylistic design, set in a suburban town. Gameplay features may include: a point system (high scores etc.), customisable character (dependent on allowing third-person camera view), a hub world (sandbox style), various levels to play (e.g. go to any dog to start a level, avoid obstacles, collect power-ups etc., randomly generated)

The motivation behind this project is due to my love of video games, and wanting to pursue making/designing them as a career after University. The project intends to provide entertainment for fellow video game lovers.

E.2 Methods/Skills

The project will require Unity (free), and potentially a modelling software - such as Blender 3D (free) - should I choose to model (and animate) my own assets. However, there are free and paid-for assets available on the Unity store should I decide not to model my own. These software applications are free, and available on macOS, Windows, and Linux operating systems, thus I have ready access to them.

I have acquired the skills needed through various modules during my time at University - including (but not limited to) Programming, Maths, Games AI Programming (due to start in term 2), and 3D Virtual Environments and Animation. As I am not entirely proficient in C#, I will need to further these skills through independent study. I will also have to learn to use Blender (modelling and animating) should I choose to make my own assets - this is very achievable through independent study, with an abundance of resources and tutorials online (likewise for Unity).

The project may potentially be made VR compatible - dependent on time constraints. Initially the project will be designed with just PC in mind, however. VR headsets (HTC Vive, and Oculus Rift) are available to loan from the University should I decide to go down this route.

E.3 Project Evaluating

The project will be evaluated on various things, such as: complexity of gameplay features, how well they were implemented, how polished the final result is (any bugs/glitches? everything working as it should? good visuals? audio? overall game design? etc.), the variety of features included, and techniques involved.

E.4 Project Planning

Christmas holidays: Play around with Unity, potential modelling in Blender, decide on gameplay features (rank in list of importance - prioritise etc., look into feasibility), design style etc.

January: Design game map, visuals, GUI, overall game design. Basic implementation in Unity.

February: Implement the technical features into the game. Basic visual design. Preliminary project report.

March: Continue technical implementation, modelling, overall visual design of the game. Start final report.

April: Finishing up the project. Final testing, focussing on report.

May: Any last minute project changes, finalising and submitting report.

E.5 References

Unity - https://unity3d.com/

Blender - https://docs.blender.org/manual/en/latest/index.html

Appendix F

Design Document

WALKIES DESIGN DOCUMENT
Final Year Project
Molly Mason
2019

F.1 Overview

'Walkies' will be a third person 3D dog walking game. It will be developed in Unity using C#, with assets modelled and animated using Blender. The name comes from the term 'walkies' - a term 'said to a dog to tell it it's time for a walk'.

Inspired by games like 'Crazy Taxi' and 'Cubefield', the game will feature a hub world, where the player can walk around a suburban town to select levels of varying difficulty. Within the levels, the player will have to walk a dog for the furthest distance possible, whilst collecting power-ups and avoiding obstacles.

The art style is inspired by games such as 'A Hat In Time', 'Overcooked', 'Glover', and 'Epic Mickey'; having a fun, simple, cartoonish, and colourful theme.

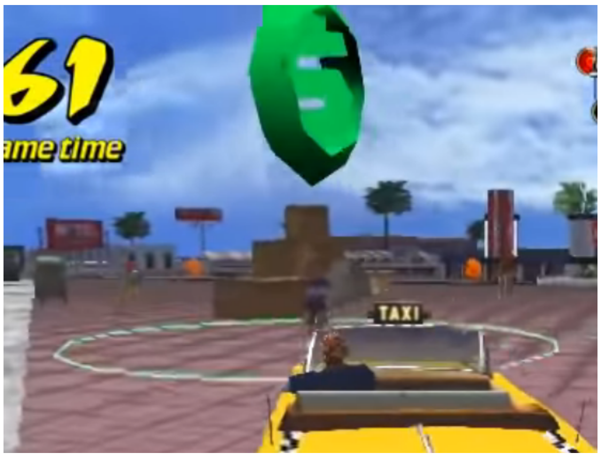
F.2 Controls

The game will be made for PC only.

KEYS	ACTION
WASD	Forward/Left/Backwards/Right avatar movement
Arrow keys	Camera view movement
Space (holding down)	Select a level when in the vicinity
ESC	Pause menu
Mouse button	Select options in the Start and Pause menus

F.3 Levels

There will be around 9 levels - 3 for each are of the world; town, suburbs, and dog park (different visuals for each area). These will be indicated through a visual circle on the ground in the hub world, with a colour to signify its difficulty - green for easy, yellow/orange for medium, red for hard - as well as an NPC with a dog standing in it. An example of this is found in Crazy Taxi:



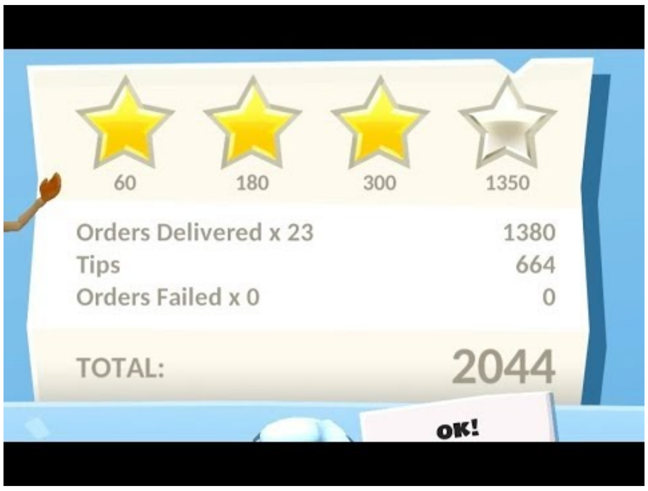
The levels will consist of the avatar moving forward at a constant speed, with the player being able to move their avatar left and right to avoid oncoming obstacles and collect power-ups - similar to Cubefield. The score is indicated through a distance number (in km) - the higher the better - that increases constantly.

The player will start off with 3 hearts of life, indicated by icons in the top right corner. Up to 2 extra lives at a time can be gained by walking through power-ups - the avatar will flash green when this happens. 1 life is lost per obstacle walked through - the avatar will flash red when this happens. The level ends when the player loses all 3 lives.

OBSTACLES	POWERUPS
Open manholes	Energy drink
Dog poop	Dog treat (bone shaped biscuit)
Trash	
Cats	

The difficulty will affect the speed of the avatar, as well as the frequency of the obstacles and power ups (hard = more obstacles, less power ups, faster starting speed). The level will get gradually faster the longer it goes on.

After a level ends, the player is greeted with a UI stating their final distance and equivalent grade - 1 to 3 stars (3 being the best) - as well as the amount of power ups collected and obstacles collided, similar to Overcooked:



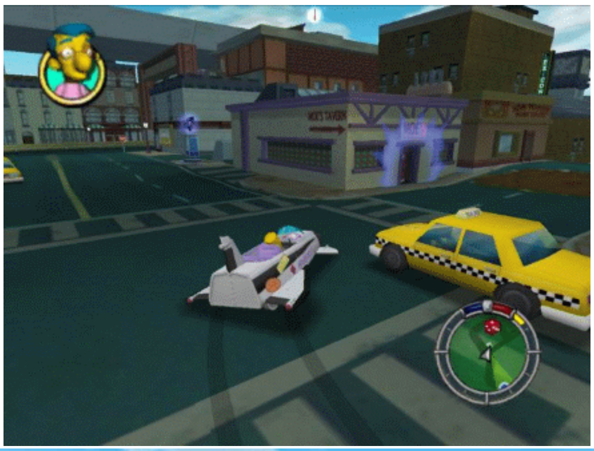
Orders Delivered x 23	1380
Tips	664
Orders Failed x 0	0
TOTAL:	2044

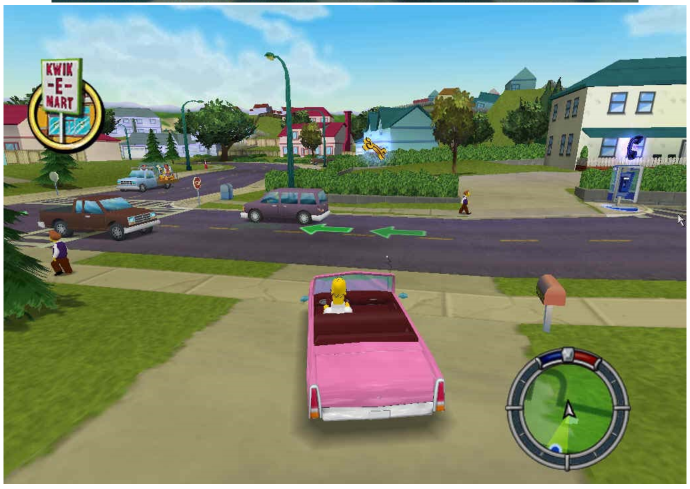
Level scores are saved automatically upon finishing a level.

F.4 Environment

The game will be set in a small town, featuring a town area, a dog park, and a suburban area. Each of the distinct areas would feature 3 levels - one easy, one medium, one hard. The level decoration corresponds to its location - the town would have you walking through town, suburbs walking through suburbs, park walking through park. None of the buildings will be enterable.

The hub world will have roads and paths to walk on. The town will have connected buildings, similar to typical town environments. The suburbs will have separated typical American suburban style one storey houses, complete with white picket fences and front gardens - colours may differ between houses for some variation. The dog park will be a typical park. Environment design will be most similar to The Simpsons Hit and Run:





Inaccessible areas will be indicated with road block assets, preventing access with invisible walls.

Set decoration around the area may include: trees, bushes, fire hydrants, trash cans, dogs, mail boxes. These cannot be interacted with.

The world may also include AI - primarily human AI, but possibly also car AI. The AI would not be interact-able with, and would roam the world independently. Human NPCs would all use the same base model, with 2 hairstyles (one for females, one for males), and colour variations for clothing and hair.

F.5 GUI

The game will have a start and pause menu. The start menu will feature a Start, Settings, and Quit option buttons. The pause menu will have a Resume, Scores, Settings, and Quit option buttons.

Within the Settings, the player can change audio settings (using sliders), as well as choose to reset the game - takes them to the start menu, and wipes all saved scores and any other saved options (including purchased hats - discussed later on in the document). Settings will save automatically upon changing them.

The pause menu comes up as an overlay on the player's game, with the game still in the background. The start menu background may be a random picture of the game, or a panning camera view of the game environment.

Level scores can be viewed via the pause menu, or I may choose to implement a 'bulletin board' feature in-game in which you can access them instead. These will be similar to the pop up shown when finishing a level, except more compact, to fit 3 levels of the area on the screen - click arrows to flip through the different areas scores (town, suburbs, park).

F.6 Other

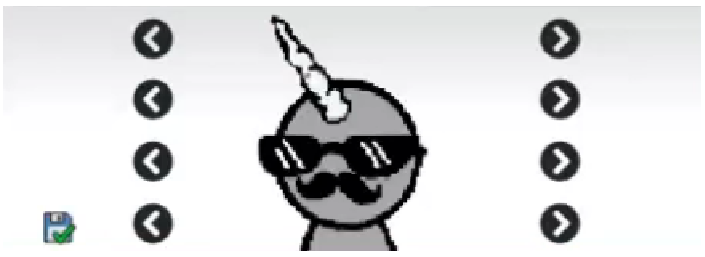
Other features that I plan to include, however some might get cut due to time restrictions.

F.6.1 Hat shop

The hat shop would be an enterable building found in the town. It would not be available (locked) until the player has completed at least one level. It would be entered by walking through a blackened out doorway, indicated by an arrow on the ground. The building would consist of a room with the display of purchasable hats, a desk with a cash register on, and a shop assistant NPC. The player can press space to go into 'hat purchasing mode' - a fixed view in the shop which allows players to click arrow keys on screen to look through the selection of hats available to buy and wear with stars earned from levels.

Each option would have a box interface with a 'buy' or 'wear' button (dependent on whether the player has bought the item), along with the price and potentially a description. The 'buy' button would be greyed out if the player does not have enough stars. Stars are cumulative, and do not get removed once they are 'spent'. A 'ca-ching' sound plays when the player buys an item.

A comparable 2D interface to this would be in the former doodly.io:



F.6.2 Day/Night system

A day/night system would feature day and night skyboxes, and a clock interface in the top left corner - simple text to indicate the time. One minute would equal one hour. The world lighting would start off brightest at noon, and gradually lower to darkness at 8pm starting from 5/6pm, again lightening from around 6am.

The time of day would only affect the skybox and world lighting - lamp posts around the world would turn on and turn off after a certain time.

F.6.3 Weather system

A weather system would give the chance for the world to cycle randomly through rain, grey cloud, and sunny weather options. This would only affect world lighting, skyboxes, and audio.

This option seems less likely to implement due to technical considerations, such as rain slowing the game down.

F.6.4 Loading screen

I may include a simple loading screen. It will likely just flash a simple 'Loading…' text with a random faded scenic background picture from the game.

Appendix G

Preliminary Project Report

Goldsmiths, University of London
IS53007D: Computing Project (2018-19)
Preliminary Project Report
Molly Mason
February 2019

G.1 Introduction

For my final year project I am making a 3D video game, using C# in Unity. The game itself is a third-person, single player, offline game, centered around 'infinite runner'-style dog walking levels, and its suburban town-style hub world. It is called 'Walkies', the name originating from the term said to a dog to tell it that it's time for a walk.

Various courses throughout my degree will help in this project, namely Programming and Algorithm modules, but also my third year 3D Environments and Animation, and Game AI Programming modules.

I chose this project due to my passion for video games, and hope to extend this into a career within the gaming industry after graduation. Additionally, the subject matter of the game is related to my personal love for dogs. My supervisor for the project is Frederic Fol Leymarie.

G.2 Aims and Objectives

My aims for this project is to produce a fully-working, high-quality, 3D video game. I aim to produce the majority of the assets used in the game myself, by making character and environment models, and a theme song - in addition to the code required. The art style for the design of assets and GUIs will follow a bright, cartoonish, fun, and simple theme - inspired by games such as 'A Hat in Time' (Gears for Breakfast, 2019), 'Overcooked' (Ghost Town Games, 2015), 'Glover' (Wikipedia.org, 2019), and 'Epic Mickey' (Disney.com, 2019).

All of the following objectives detailed in this game design outline will help to meet my aim of producing a high-quality 3D video game.

G.2.1 Controls

The game will be made for PC only.

KEYS	ACTION
WASD	Forward/Left/Backwards/Right avatar movement
Arrow keys	Camera view movement
Space (holding down)	Select a level when in the vicinity
ESC	Pause menu
Mouse button	Select options in the Start and Pause menus

G.2.2 Game levels

My game 'Walkies' will feature 9 levels - 3 for each area of the world; town, suburbs, and dog park (different visuals for each area). These will be indicated through a visual circle on the ground in the hub world, with a colour to signify its difficulty - green for easy, yellow/orange for medium, red for hard - as well as an NPC with a dog standing in it.

The levels will consist of the avatar moving forward at a constant speed, with the player being able to move their avatar left and right to avoid randomly-spawned oncoming obstacles and collect power-ups - similar to 'Cubefield' (Cubefield, 2019). The score is indicated through a distance number (in km) - the higher the better - that increases constantly.

The player will start off with 3 hearts of life, indicated by icons in the top right corner. Up to 2 extra lives at a time can be gained by walking through power-ups - the avatar will flash green when this happens. 1 life is lost per obstacle walked through - the avatar will flash red when this happens. The level ends when the player loses all 3 lives.

OBSTACLES	POWERUPS
Open manholes	Energy drink
Dog poop	Dog treat (bone shaped biscuit)
Trash	
Cats	

The difficulty will affect the speed of the avatar, as well as the frequency of the obstacles and power ups (hard = more obstacles, less power ups, faster starting speed). The level will get gradually faster the longer it goes on. After a level ends, the player is greeted with a UI stating their final distance and equivalent grade - 1 to 3 stars (3 being the best) - as well as the amount of power ups collected and obstacles collided. Level scores are saved automatically upon finishing a level.

G.2.3 Game Environment

The game will be set in a small town, featuring a town area, a dog park, and a suburban area. Each of the distinct areas would feature 3 levels - one easy, one medium, one hard. The level decoration corresponds to its location - the town would have you walking through town, suburbs walking through suburbs, park walking through park. None of the buildings will be enterable.

The hub world will have roads and paths to walk on. The town will have connected buildings, similar to typical town environments. The suburbs will have separated typical American suburban style one storey houses, complete with white picket fences and front gardens - colours may differ between houses for some variation. The dog park will be a typical park.

Inaccessible areas will be indicated with road block assets, preventing access with invisible walls. Set decoration around the area will include: trees, bushes, fire hydrants, trash cans, dogs, mail boxes. These cannot be interacted with.

The world will include basic human AI. Human NPCs would all use the same base model, with 2 hairstyles (one for females, one for males), and colour variations for clothing and hair.

G.2.4 GUI

The game will have a start and pause menu. The start menu will feature a Start, Settings, and Quit option buttons. The pause menu will have a Resume, Scores, Settings, and Quit option buttons.

Within the Settings, the player can change audio settings (using sliders), as well as choose to reset the game - takes them to the start menu, and wipes all saved scores and any other saved options. Settings will save automatically upon changing them.

The pause menu comes up as an overlay on the player's game, with the game still in the background. The start menu background will be a custom animated background of blue sky scenery. A plain loading screen will be visible whilst levels are loading.

Level scores will be viewed via a 'bulletin board' feature in-game. This will be similar to the pop up shown when finishing a level, except more compact, to fit 3 levels of the area on the screen - click arrows to flip through the different areas scores (town, suburbs, park).

G.2.5 Other

A feature to incentivise players to replay levels for the best score is that of the hat shop. This will be an enterable building found in the town. It will not be available (locked) until the player has completed at least one level, and can be entered by walking through a blackened out doorway, indicated by an arrow on the ground.

The building will consist of a room with the display of purchasable hats, a desk with a cash register on, and a shop assistant NPC. The player can press space to go into 'hat purchasing mode' - a fixed view in the shop which allows players to click arrow keys on screen to look through the selection of hats available to buy and wear with stars earned from levels.

Each option would have a box interface with a 'buy' or 'wear' button (dependent on whether the player has bought the item), along with the price and a humorous description. The 'buy' button will be greyed out if the player does not have enough stars. Stars are cumulative, and do not get removed once they are 'spent'. A 'caching' sound plays when the player buys an item.

Dependent on time constraints, I may look into implementing a Day and Night system, a Weather system.

G.2.6 Summary

To summarise, the main deliverables needed for my project can be split up into the following categories:

- Hub world creation
- Level creation
- Artificial Intelligence (AI) implementation
- GUI creation
- Asset modelling

G.3 Methods

In order to achieve my aims detailed in the previous section, I will be using the video game engine Unity (Unity Technologies, 2019) to make the game, writing code in the language C#. 3D assets - such as character and environment models - will be modelled using the modelling software Blender. For the game design, I have already utilised a game design document template to help me plan out my project in detail.

I intend to use Unity as it is a free software, widely available for use. Its main language, C#, is very similar to Java - which I learned in my year 2 programming module, and thus learning it shouldn't be too hard. Blender (Blender, 2019) will be used for modelling as it is also a widely available free software. Both of these programs have multitudes of tutorials available online, to help me through the learning process - as I will be learning as I go on this project.

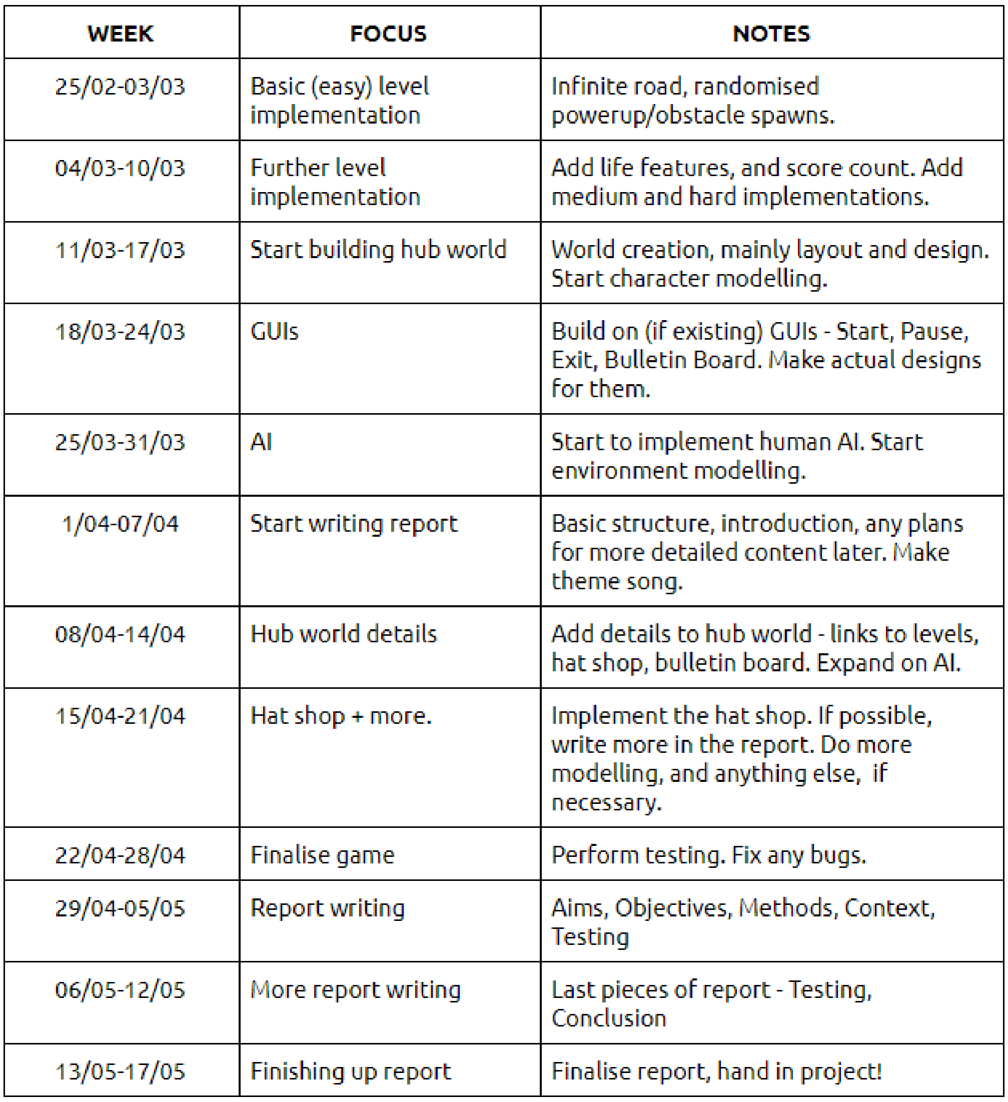
For any graphic design needed for GUIs, I will likely be using a mixture of free graphics editor software GIMP (GIMP, 2019), and free vector graphics software Inkscape (Inkscape, 2019). I will be using these as I am already familiar with them, in addition to them also being free and widely available.

Git (Git, 2019) version control may also be used to backup my project code, with Github (Github, Inc., 2019). For any sounds, I will likely use freesound (Freesound, 2019), for free to use sound effects.

I plan to focus on AI in the project, once the base features are all completed. This will use a mixture of finite state machines and behaviour trees for a suitable level of complexity in relation to the rest of the project design and time constraints. These methods will also be suitable due to having learnt them recently in my Game AI Programming module.

G.4 Project Plan

Here is my plan detailing all deliverables and goals I plan to reach and when, from the past Christmas, to now, up until the final due date on May 17th:



WEEK	FOCUS	NOTES
25/02-03/03	Basic (easy) level implementation	Infinite road, randomised powerup/obstacle spawns.
04/03-10/03	Further level implementation	Add life features, and score count. Add medium and hard implementations.
11/03-17/03	Start building hub world	World creation, mainly layout and design. Start character modelling.
18/03-24/03	GUIs	Build on (if existing) GUIs - Start, Pause, Exit, Bulletin Board. Make actual designs for them.
25/03-31/03	AI	Start to implement human AI. Start environment modelling.
1/04-07/04	Start writing report	Basic structure, introduction, any plans for more detailed content later. Make theme song.
08/04-14/04	Hub world details	Add details to hub world - links to levels, hat shop, bulletin board. Expand on AI.
15/04-21/04	Hat shop + more.	Implement the hat shop. If possible, write more in the report. Do more modelling, and anything else, if necessary.
22/04-28/04	Finalise game	Perform testing. Fix any bugs.
29/04-05/05	Report writing	Aims, Objectives, Methods, Context, Testing
06/05-12/05	More report writing	Last pieces of report - Testing, Conclusion
13/05-17/05	Finishing up report	Finalise report, hand in project!

G.5 Progress to Date

So far, my work has primarily consisted of planning out the game design, assets, GUIs, and world layout [Appendix A].

Additionally, as a lot of the methods used in this project are new to me, I have mainly been teaching myself the necessary knowledge in which to program my game and model the assets. So far, I do not have any concrete finalised work, instead just test models and environments to feel confident enough in my skills.

This has included Unity's Official Tutorials (Unity Technologies, 2019), as well as following Darrin Lile's Blender Character modelling series on Youtube (Lile, 2014). In addition to just character modelling, I have learned about UV mapping, character rigging, character animation, and how to export and import Blender models into the Unity environment.

Background research to help me visualise and plan the overall game design has been conducted by looking into, and gaining inspiration from games such as:

- The Simpsons Hit and Run (town design, level pickup design) (Wikipedia.org, 2019)

- Overcooked (character design)
- A Hat in Time (overall art style and tone)

This has consisted of watching gameplay - what makes it entertaining? - as well as observing level and character design - what makes it so charming? I did look into other games, such as 'Yooka-Laylee' (Playtonic Games, 2019), 'Super Mario Odyssey' (Nintendo, 2019), and 'Slime Rancher' (Monomi Park, 2019); however those listed felt more appropriate to my project. This background research led to the creation of a design document for 'Walkies' - in which I received feedback about my envisioned project from my supervisor, and included details from it within this report.

Recently I have started to make a prototype for the levels, involving multiple plane objects that move (in a conveyor-like fashion) once the player reaches a certain point on the following plane - giving the illusion of an infinite road/path. This has proven to be harder than I initially anticipated, already setting my planned progress back a bit.

Testing the assets I may need in Unity also led me to discover that I will need to alter Unity's standard third-person controller asset (Unity Technologies, 2019) to fit my control design. Currently, Unity's third-person controller script moves the player using WASD keys in addition to arrow keys, as opposed to my planned WASD for movement, arrow keys for camera.

G.6 Planned Work

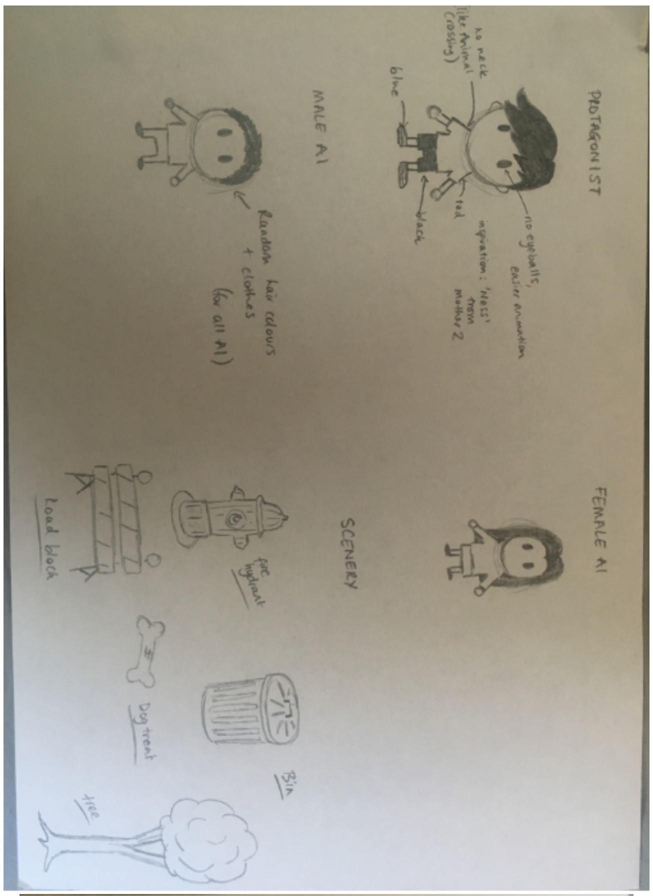
Right now, the project is in the very early stages of prototype implementation. I intend to continue this prototype implementation, aiming to finish the level creation by the end of February at the latest. This consists of getting the infinite road to work, as well as randomly spawning a mixture of power-up objects and obstacle objects. Lastly, the life and score system will be implemented. It is likely that I will not add any styling to the environment until I'm at a further stage in the project.

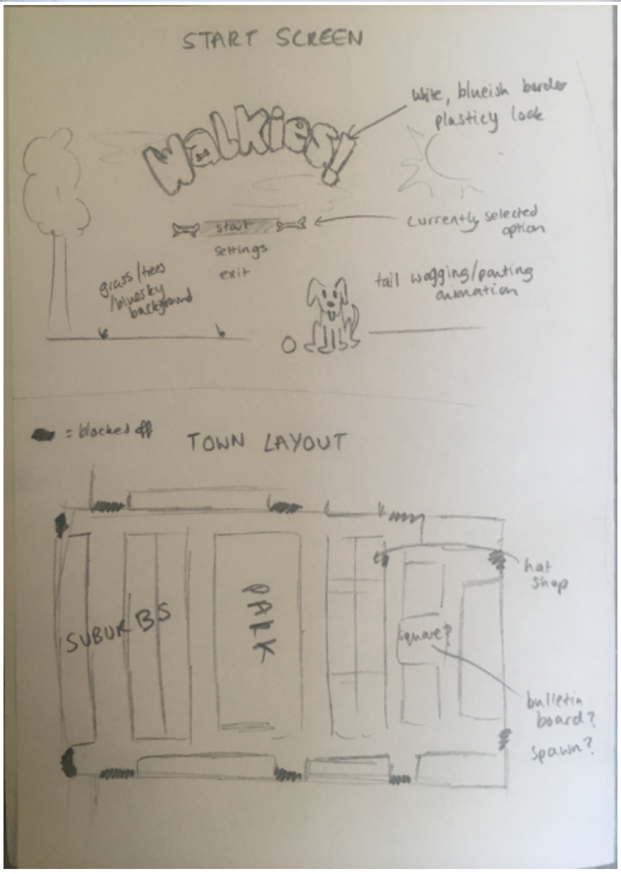
Getting this main hurdle out of the way will allow me to focus on other aspects of the project, namely building the hub world, implementing some AI, and adding the GUIs. Lower priority goals are that of the hat shop, and overall modelling and styling of the game - these can be left until the harder parts are out of the way.

G.7 Appendix

G.7.1 Appendix A

Design Sketches


PROTAGONIST
no eyeballs, easier animation
inspiration: 'Ness' from Mother 2
no neck (like Animal Crossing)
blue
red
black
MALE AI
Random hair colours + clothes (for all AI)
FEMALE AI
SCENERY
fire hydrant
Bin
road block
Dog treat
tree


START SCREEN
Walkies!
White, blueish border plasticy look
start
settings
exit
currently selected option
grass/trees/blue sky background
tail wagging/panting animation
= blocked off
TOWN LAYOUT
SUBURBS
PARK
hat shop
square?
bulletin board? spawn?

G.8 References

Blender. 2019. blender.org - Home of the Blender project - Free and Open 3D Creation Software [Online]. Available at: https://www.blender.org/ [Accessed: 22 February 2019].

Cubefield.org.uk. 2019. Cubefield - Play Online Free! [Online]. Available at: http://www.cubefield.org.uk/ [Accessed: 22 February 2019].

Disney.com. 2019. Disney Epic Mickey | Disney LOL [Online]. https://lol.disney.com/games/disney-epic-mickey-video-game [Accessed: 22 February 2019].

Freesound. 2019. Freesound - freesound [Online]. Available at: https://freesound.org/ [Accessed: 22 February 2019].

GIMP. 2019. GIMP - GNU Image Manipulation Program [Online]. Available at: https://www.gimp.org/ [Accessed: 22 February 2019].

Gears for Breakfast. 2019. A Hat in Time - Cute-as-heck 3D Platformer! [Online]. Available at: http://hatintime.com/ [Accessed: 22 February 2019].

Ghost Town Games. 2015. Overcooked [Online]. Available at: http://www.ghosttowngames.com/overcooked/ [Accessed: 22 February 2019].

Git, 2019. Git [Online]. Available at: https://git-scm.com/ [Accessed: 22 February 2019].

GitHub, Inc. 2019. The world's leading software development platform - Github [Online]. Available at: https://github.com/ [Accessed: 22 February 2019].

Inkscape. 2019. Draw Freely | Inkscape [Online]. Available at: https://inkscape.org/ [Accessed: 22 February 2019].

Lile, D. 2014. Blender Character Modeling 1 of 10 [Online]. Available at: https://www.youtube.com/watch?v=0QT1GNMevfc [Accessed: 22 February 2019].

Monomi Park. 2019. Slime Rancher [Online]. Available at: http://slimerancher.com/ [Accessed: 22 February 2019].

Nintendo. 2019. Super Mario Odyssey^{TM} for the Nintendo Switch^{TM} home gaming system - Official Game Site [Online]. Available at: https://supermario.nintendo.com/ [Accessed: 22 February 2019].

Playtonic Games. 2019. Yooka-Laylee - Playtonic Games [Online]. Available at: https://www.playtonicgames.com/games/yooka-laylee/ [Accessed: 22 February 2019].

Unity Technologies. 2019. Unity Learn Tutorials [Online]. Available at: https://unity3d.com/learn/tutorials [Accessed: 22 February 2019].

Unity Technologies. 2019. Standard Assets - Asset Store [Online]. Available at: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-32351 [Accessed: 22 February 2019].

Unity Technologies. 2019. Unity [Online]. Available at: https://unity3d.com/ [Accessed: 22 February 2019].

Wikipedia.org. 2019. Glover (video game) - Wikipedia [Online]. Available at: https://en.wikipedia.org/wiki/Glover_(video_game) [Accessed: 22 February 2019].

Wikipedia.org. 2019. The Simpsons: Hit & Run - Wikipedia [Online]. Available at: https://en.wikipedia.org/wiki/The_Simpsons:_Hit_%26_Run [Accessed: 22 February 2019].