

Richelle Acheampong

AstroSprint

Table Of Contents

<u>1 - Analysis</u>	8
<u>1.1 - Introduction</u>	8
<u>1.2 - Stakeholders</u>	10
<u>1.3 - Research</u>	11
<u>1.3.1 - Existing Solutions</u>	11
<u>1.3.2 - Stakeholder Questionnaire</u>	18
<u>1.4 - Hardware & Software Requirements</u>	20
<u>1.4.1 - Developer Requirements</u>	20
<u>1.4.2 - User Requirements</u>	21
<u>1.5 - Success Criteria</u>	21
<u>1.5.1 - Menus & UI Criteria</u>	21
<u>1.5.2 - Gameplay Criteria</u>	26
<u>1.5.3 - External Database Criteria</u>	31
<u>2 - Design</u>	34
<u>2.1 - Menus Flowchart</u>	34
<u>2.2 - Game Decomposition</u>	35
<u>2.2.1 - Main Decomposition Diagram</u>	35
<u>2.2.1.1 - Alternative Decomposition</u>	35
<u>2.2.2 - Login System Decomposition</u>	36
<u>2.2.3 - Sign Up System</u>	37
<u>2.2.4 - Easy Mode Decomposition</u>	37
<u>2.2.5 - Hard Mode Decomposition</u>	39
<u>2.2.6 - Abilities Menu Decomposition</u>	40
<u>2.2.7 - Character Menu Decomposition</u>	40
<u>2.2.8 - Pause Menu Decomposition</u>	42
<u>2.2.9 - Game Summary Decomposition</u>	43
<u>2.3 - Main Menu Design</u>	43
<u>2.3.1 - Initial Designs</u>	43
<u>2.3.2 - Stakeholder Review</u>	44
<u>2.3.3 - Evaluation of Review</u>	46
<u>2.3.4 - Final Design</u>	46
<u>2.4 - Menu Designs & Icons</u>	47
<u>2.4.1 - Sign Up Menu</u>	47
<u>2.4.2 - Login Menu</u>	48
<u>2.4.3 - Play Menu</u>	49
<u>2.4.4 - Tutorial Screen</u>	49
<u>2.4.5 - Abilities Menu</u>	50
<u>2.4.6 - Characters Menu</u>	51
<u>2.4.7 - Pause Menu</u>	51
<u>2.4.8 - Game Summary Screen</u>	52
<u>2.4.9 - Icons</u>	52
<u>2.5 - Key Process Flowcharts</u>	55
<u>2.5.1 - Movement Flowchart</u>	55
<u>2.5.2 - Jumping Subroutine Flowchart</u>	56

<u>2.5.3 - Game Pause Flowchart</u>	57
<u>2.5.4 - Coin Generation Flowchart</u>	63
<u>2.5.5 - Obstacle Generation Flowchart</u>	64
<u>2.5.6 - Player Death Subroutine Flowchart</u>	65
<u>2.5.7 - Scoring System Flowchart</u>	66
<u>2.5.8 - Sign Up Flowchart</u>	67
<u>2.5.9 - Log In Flowchart</u>	73
<u>2.5.10 - Flying (Power Up) Flowchart</u>	78
<u>2.5.11 - Half Score (Power Down) Flowchart</u>	79
<u>2.5.12 - Blurry Screen (Power Down) Flowchart</u>	80
<u>2.5.12.1 - Pseudocode Representation</u>	80
<u>2.5.12.2 - Algorithmic Representation</u>	80
<u>2.5.13 - Darkness (Power Down) Flowchart</u>	82
<u>2.5.12.1 - Pseudocode Representation</u>	82
<u>2.5.12.2 - Algorithmic Representation</u>	82
<u>2.5.14 - Inverted Colours (Power Down) Flowchart</u>	84
<u>2.5.15 - Character Selection Flowchart</u>	85
<u>2.5.16 - Abilities Upgrade Flowchart</u>	86
<u>2.5.17 - Displaying Ability Levels Flowcharts</u>	89
<u>2.5.18 - Purchasing Ability Upgrade Subroutine Flowcharts</u>	92
<u>2.5.19 - Generating Power Ups Flowchart</u>	94
<u>2.5.20 - Generating Power Downs Flowchart</u>	95
<u>2.5.21 - Game Timer Flowchart</u>	96
<u>2.5.21.1 - Pseudocode Representation</u>	96
<u>2.5.21.2 - Algorithmic Representation</u>	96
<u>2.6 - Key Variables</u>	97
<u>2.6.1 - Sign Up Key Variables</u>	97
<u>2.6.2 - Login Key Variables</u>	98
<u>2.6.3 - Play Menu Key Variables</u>	99
<u>2.6.4 - Tutorial Key Variables</u>	99
<u>2.6.5 - Abilities Menu Key Variables</u>	99
<u>2.6.6 - Character Menu Key Variables</u>	101
<u>2.6.7 - Pause Menu Key Variables</u>	101
<u>2.6.8 - Easy Mode Key Variables</u>	102
<u>2.6.9 - Hard Mode Key Variables</u>	104
<u>2.6.10 - Game Summary Key Variables</u>	106
<u>2.7 - Key Data Structures</u>	107
<u>2.8 - Input Field Processes</u>	107
<u>2.8.1 - User Sign Up Details</u>	107
<u>2.8.2 - User Login Details</u>	109
<u>2.9 - Database</u>	110
<u>2.9.1 - The Need for a Database</u>	110
<u>2.9.2 - Database Implementation</u>	111
<u>2.10 - Usability Features</u>	111
<u>2.10.1 - Colour</u>	111
<u>2.10.2 - Layout</u>	114

<u>2.10.3 - Text</u>	114
<u>2.10.4 - User-Friendly Approach</u>	115
<u>2.11 - Iterative Development Test Planning</u>	115
<u>2.11.1 - Sign Up Module Testing</u>	115
<u>2.11.2 - Login Module Testing</u>	118
<u>2.11.3 - Character Menu Testing</u>	119
<u>2.11.4 - Tutorial Screen Testing</u>	120
<u>2.11.5 - Abilities Menu Testing</u>	120
<u>2.11.6 - Gameplay Testing</u>	122
<u>2.11.7 - Game Summary Testing</u>	126
<u>2.12 - Post Development Test Planning</u>	126
<u>3 - First Iteration Development (AstroSprint v1.0)</u>	138
<u>3.1 - AstroSprint v1.0 Summary</u>	138
<u>3.2 - Main Menu Development</u>	138
<u>3.2.1 - Main Menu Development Interface</u>	138
<u>3.2.2 - Main Menu Code</u>	141
<u>3.3 - Sign Up Menu Development</u>	143
<u>3.3.1 - Sign Up Menu Game Objects</u>	143
<u>3.3.2 - Sign Up Menu Code</u>	144
<u>3.3.3 - Sign Up Menu Iterative Testing</u>	150
<u>3.3.3.1 - Test Table</u>	150
<u>3.3.3.2 - Test Evidence</u>	153
<u>3.3.4 - Review of Sign Up Menu Development</u>	155
<u>3.4 - Login Menu Development</u>	156
<u>3.4.1 - Login Menu Game Objects</u>	156
<u>3.4.2 - Login Menu Code</u>	157
<u>3.4.3 - Login Menu Iterative Testing</u>	161
<u>3.4.3.1 - Test Table</u>	161
<u>3.4.3.2 - Test Evidence</u>	162
<u>3.4.4 - Review of Login Menu Development</u>	163
<u>4 - Second Iteration Development (AstroSprint v2.0)</u>	165
<u>4.1 - AstroSprint v2.0 Summary</u>	165
<u>4.2 - Play Menu Development</u>	165
<u>4.2.1 - Play Menu Game Objects</u>	165
<u>4.2.2 - Play Menu Code</u>	165
<u>4.2.3 - Review of Play Menu Development</u>	167
<u>4.3 - Character Menu Development</u>	167
<u>4.3.1 - Character Menu Game Objects</u>	168
<u>4.3.2 - Character Menu Code</u>	168
<u>4.3.3 - Character Menu Iterative Testing</u>	171
<u>4.3.3.1 - Test Table</u>	171
<u>4.3.3.2 - Test Evidence</u>	173
<u>4.3.4 - Review of Character Menu Development</u>	175
<u>4.4 - Abilities Menu Development</u>	176
<u>4.4.1 - Abilities Menu Game Objects</u>	176
<u>4.4.2 - Abilities Menu Code</u>	177

<u>4.4.3 - Abilities Menu Iterative Testing</u>	183
<u>4.4.3.1 - Test Table</u>	183
<u>4.4.3.2 - Test Evidence</u>	184
<u>4.4.4 - Review of Abilities Menu Development</u>	185
<u>4.5 - Tutorial Screen Development</u>	186
<u>4.5.1 - Tutorial Screen Game Objects</u>	186
<u>4.5.2 - Tutorial Screen Code</u>	186
<u>4.5.3 - Tutorial Screen Testing</u>	187
<u>4.5.3.1 - Test Table</u>	187
<u>4.5.3.2 - Test Evidence</u>	187
<u>4.5.4 - Review of Tutorial Screen Development</u>	187
<u>5 - Third Iteration Development (AstroSprint v3.0)</u>	189
<u>5.1 - AstroSprint v3.0 Summary</u>	189
<u>5.2 - Easy Mode Development</u>	189
<u>5.2.1 - Easy Mode Game Objects</u>	189
<u>4.6.2 - Easy Mode Code</u>	190
<u>5.2.2.1 - Coin Spawner Script</u>	190
<u>5.2.2.2 - Coin Prefab Script</u>	191
<u>5.2.2.3 - Foreground Movement Script</u>	192
<u>5.2.2.4 - Obstacle Spawner Script</u>	194
<u>5.2.2.5 - Obstacle Prefab Script</u>	195
<u>5.2.2.6 - Play Manager Script</u>	197
<u>5.2.2.7 - Power Up Spawner Script</u>	198
<u>5.2.2.8 - Double Jump Script</u>	200
<u>5.2.2.9 - Double Coins Script</u>	202
<u>5.2.2.10 - Double Score Script</u>	204
<u>5.2.2.11 - Flying Script</u>	205
<u>5.2.2.12 - Game Pause Script</u>	207
<u>5.2.3 - Easy Mode Iterative Testing</u>	209
<u>5.2.3.1 - Test Table</u>	209
<u>5.2.3.2 - Test Evidence</u>	211
<u>5.2.4 - Review of Easy Mode Development</u>	217
<u>5.3 - Game Summary Development</u>	218
<u>5.3.1 - Game Summary Game Objects</u>	218
<u>5.3.2 - Game Summary Code</u>	219
<u>5.3.3 - Game Summary Iterative Testing</u>	222
<u>5.3.3.1 - Test Table</u>	222
<u>5.3.3.2 - Test Evidence</u>	222
<u>5.3.4 - Review of Game Summary Development</u>	224
<u>6.1 - Feedback on AstroSprint v4.0</u>	225
<u>6.1.1 - Primary User Introduction & Feedback on v4.0</u>	225
<u>6.1.1.1 - Primary User Introduction</u>	225
<u>6.1.1.2 - Primary User Feedback</u>	225
<u>6.1.2 - Developer Feedback on v4.0</u>	226
<u>6.1.2.1 - Video Evidence</u>	226
<u>6.1.2.2 - Developer Issues</u>	226

<u>6.1.2.3 - Shortlisted Features To Be Implemented</u>	228
<u>6.2 - AstroSprint v4.0 Success Criteria</u>	228
<u>6.3 - AstroSprint v4.0 Development</u>	231
<u>6.3.1 - AstroSprint v4.0 Changes</u>	231
<u>6.3.1.1 - Changing Background (S.C. No. 1)</u>	231
<u>6.3.1.2 - Sign Up Menu Font (S.C. No. 2)</u>	232
<u>6.3.1.3 - Power Up Spawn Interval (S.C. No. 3)</u>	234
<u>6.3.1.4 - Obstacle Spawn Interval (S.C. No. 4)</u>	234
<u>6.3.1.5 - Changing Pause Menu Function (S.C. No. 5)</u>	234
<u>6.3.1.6 - Obstacle Falling Physics (S.C. No. 6)</u>	236
<u>6.3.1.7 - Obstacle Spawn Height (S.C. No. 7)</u>	237
<u>6.3.1.8 - Obstacle Fall Speed (S.C. No. 8)</u>	237
<u>6.3.1.9 - Removal of Hard Mode (S.C. No. 9)</u>	238
<u>6.3.1.10 - Renaming of Easy Mode (S.C. No. 10)</u>	239
<u>6.3.1.11 - Rearrangement of Play Menu (S.C. No. 11)</u>	240
<u>6.3.1.12 - Frequency of Coin Balance (S.C. No. 12)</u>	241
<u>6.3.1.13 - Renaming of Tutorial Screen (S.C. No. 13)</u>	242
<u>6.3.2 - AstroSprint v4.0 Testing</u>	244
<u>6.3.2.3 - Test Planning</u>	244
<u>6.3.2.2 - Test Table</u>	245
<u>6.3.2.3 - Test Evidence</u>	247
<u>6.3.3 - Review of AstroSprint v4.0 Development</u>	247
<u>7.1 - Post Development Testing</u>	248
<u>7.1.1 - Further Evidence</u>	258
<u>7.2 - Usability Testing</u>	261
<u>7.2.1 - Main Menu Usability</u>	261
<u>7.2.1.1 - Evidence of Features</u>	261
<u>7.2.1.2 - Primary User Feedback</u>	262
<u>7.2.1.3 - Suggested Features</u>	263
<u>7.2.2 - Login Menu Usability</u>	264
<u>7.2.2.1 - Evidence of Features</u>	264
<u>7.2.2.2 - Feedback from Primary User</u>	264
<u>7.2.2.3 - Suggested Features</u>	265
<u>7.2.3 - Sign Up Menu Usability</u>	266
<u>7.2.3.1 - Evidence of Features</u>	266
<u>7.2.3.2 - Feedback from Primary User</u>	266
<u>7.2.3.3 - Suggested Features</u>	268
<u>7.2.4 - Play Menu Usability</u>	268
<u>7.2.4.1 - Evidence of Features</u>	268
<u>7.2.4.2 - Feedback from Primary User</u>	268
<u>7.2.4.3 - Suggested Features</u>	269
<u>7.2.5 - Controls Screen Usability</u>	269
<u>7.2.5.1 - Evidence of Features</u>	270
<u>7.2.5.2 - Feedback from Primary User</u>	270
<u>7.2.5.3 - Suggested Features</u>	271
<u>7.2.6 - Character Menu Usability</u>	271

<u>7.2.6.1 - Evidence of Features</u>	271
<u>7.2.6.2 - Feedback from Primary User</u>	271
<u>7.2.6.3 - Suggested Features</u>	272
<u>7.2.7 - Abilities Menu Usability</u>	273
<u>7.2.7.1 - Evidence of Features</u>	273
<u>7.2.7.2 - Feedback from Primary User</u>	273
<u>7.2.7.3 - Suggested Features</u>	274
<u>7.2.8 - Gameplay Usability</u>	275
<u>7.2.8.1 - Evidence of Features</u>	275
<u>7.2.8.2 - Feedback from Primary User</u>	275
<u>7.2.8.3 - Suggested Features</u>	277
<u>7.2.9 - Pause Menu Usability</u>	277
<u>7.2.9.1 - Evidence of Features</u>	277
<u>7.2.9.2 - Feedback from Primary User</u>	277
<u>7.2.9.3 - Suggested Features</u>	278
<u>7.2.10 - Game Summary Menu Usability</u>	278
<u>7.2.10.1 - Evidence of Features</u>	279
<u>7.2.10.2 - Feedback from Primary User</u>	279
<u>7.2.10.3 - Suggested Features</u>	280
<u>7.3 - Fulfillment of Success Criteria</u>	280
<u>7.3.1 - Menus & User Interface</u>	280
<u>7.3.2 - Gameplay</u>	287
<u>7.3.3 - External Database</u>	294
<u>7.4 - Opportunities for Further Development</u>	298
<u>7.4.1 - Game Functionality</u>	298
<u>7.4.2 - Usability Features</u>	301
<u>7.5 - Maintenance & Limitations</u>	304
<u>7.5.1 - Maintenance Issues</u>	304
<u>7.5.2 - Limitations of Solution</u>	304
<u>7.6 - Opportunities to Minimise Limitations</u>	306
<u>7.6.1 - Potential Development to Deal with Limitations</u>	306
<u>7.6.2 - Potential Development to Deal with Potential Changes</u>	308
<u>8 - Bibliography</u>	308

1 - Analysis

1.1 - Introduction

Every single year, tens of thousands of games are released into the market, available to download and play. In 2023, on the platform Steam alone, 14,532 games were released (Source: [GameRant](#)). Yet thousands of games continually go unnoticed, reaching very few users and receiving little to no downloads. Users want riveting, awe-inspiring, innovative content to invigorate their simple day to day lives. The online game market needs individuality and revolution.

For my project, I aim to create a distinctive endless runner game, a type of platform game in which the player runs for an infinite amount of time, avoiding obstacles in order to obtain as high a score as possible. I believe that this will be an effective solution to the problem of there being too many boring games. This is because, unlike games with levels or a story, an endless game never has any end, therefore, the entertainment from it will also never end.

This is also strengthened by the self-competition that comes with the game, as the aim is to always beat your previous score, so the entertainment and the competitiveness of the game never ends. AstroSprint will also include power-ups and other upgrades that can enhance the playing experience, and prevent the game from having a monotonous nature. The aim of my project is to revolutionise the online game market and remedy the monotony of the currently available games and the tedium of day to day life. For a more clear view on how to enhance the online gaming experience, I am planning to research previous similar games and people's experiences with them to surpass them in terms of entertainment and enjoyment.

Features That Make the Problem Solvable by Computational Methods

An endless runner game is one that would require a repeating sequence of a background and objects to periodically appear for the player to avoid. This repeating sequence of background I believe could be solved computationally through the use of iteration, such as count controlled loops for repeating sections of background and condition controlled loops to end the loop when the game is ended by the player.

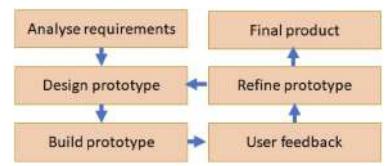
Another programming construct, the use of selection, would also be utilised within AstroSprint. This would be most obvious within the menu of the game, where the player selects what they wish to do, such as accessing the settings, playing the game, or checking previous scores. The use of programming constructs being integrated throughout the entirety of the game makes it able to be solved by computational methods.

The programming construct of sequence would also be used within AstroSprint. This would be prevalent when the game is first opened, as there will have to be a sequence of instructions in the beginning in order to have the user interface (game menu) load and have the user be able to interact with the program and play the game.

This is amenable to a computational approach, as these programming constructs are core aspects to ensure AstroSprint's functionality.

Data handling will also be used within AstroSprint. To store high scores, the user's score within the game will be recorded within a database, compared to previous scores to find the highest, and kept in case the user wants to access it and view their previous scores and current highscore. This data will be collected through the user interface, and will be available to the user on-demand.

The software development methodology of Agile Development will also be used within AstroSprint. The focus on iteration and user feedback within this methodology is something I believe will be extremely beneficial to raising the quality and standards of AstroSprint. This is something that I believe is a feature that makes AstroSprint able to be solved computationally.



The concept of 'Divide and conquer' will also be used within the development of AstroSprint. This is because the game I am proposing to create is a very large project, with smaller aspects (menus, individual game modes, high score system) that can be modularised and split into subproblems that can each be encountered concurrently and brought together at the end of the development of all of them to complete my final solution. This is a easily computerised method and therefore is another aspect of AstroSprint that makes it able to be solved by computational methods

Limitations of Proposed Solution

One limitation of my proposed solution may end up being repetitive and monotonous if played for a long period of time, or repeatedly. This is because the game is essentially repeating itself many times over, which for players who play the game often, they will get used to the gameplay and begin to find it dull, or lacking in excitement.

Another limitation may be the fact that some players may find it frustrating and stop playing the game if they find it too difficult to beat their previous high score. This is because, after trying many times over, they may give up because the game is too difficult, and stop playing the game entirely.

Another limitation may be the fact that endless runner games rely on fast reflexes, so the game may prove to not be as fun for those with slower reflexes or distractions while playing the game. This is because these people

won't be able to avoid the obstacles being presented in the game, making it so they continuously lose, which may lead them to being frustrated and quitting the game.

Another limitation of my solution is that it is extremely difficult to accommodate some physical disabilities within AstroSprint, making it less accessible for certain groups of people. This is because the game requires fast visual processing, as well as fast movements. Fast visual processing would not be possible for those with visual impairments, or those with visual information processing difficulties. Fast moments would be difficult for those who suffer from movement limiting disabilities, such as dyspraxia, which is a condition affecting physical coordination and fine motor skills.

Another limitation to the creation of my solution is time constraints. Many successful games, especially endless runners, have large teams working on the development full time (35+ hours a week) with a much larger time frame. I am only 1 person working on a game and am unable to work on it full time due to other educational commitments, as well as only having until the end of my A-Level course to complete this.

1.2 - Stakeholders

The stakeholders I have identified for this project are teenagers and young adults ages 13-18. I have chosen this group of people as stakeholders as this age group is usually within formal education, and so do not have an extensive amount of time to be engaging within an online game. My endless runner game would be an appropriate solution to their problem of the current game market being boring for many reasons:

- Due to the versatility of game duration, this game can easily fit into the busy schedules of people in this age range. The game can easily be played for short periods of time, such as on the way to an educational institution or between lessons, or can be played for longer periods of time during their downtimes, such as break times, or on weekends. The spread of durations that the game can be played for is much more suitable for this demographic, as they can adjust it to their own personal schedules and preferences, and begin or end the game quickly, whenever they need to.
- Another reason why this game is an appropriate solution is because of the concept of anticipation within the game. The unknown nature of the powerups and the next obstacles coming up within the game makes the game unpredictable and exhilarating for the player, making it captivating for a demographic of people who usually have a shorter attention span than other age groups.

My stakeholders will make use of my proposed solution by playing the game on their personal computers and being endlessly entertained by the game for as long as they are able to, easily exiting the game and being able to implement convenient entertainment within their busy lives.

1.3 - Research

1.3.1 - Existing Solutions

Subway Surfers

This is a single player endless runner mobile game, developed by SYBO and Kiloo within the Unity game engine. The aim of the game is to play as a teenager running on train tracks in order to outrun a train inspector, while dodging obstacles such as trains & barriers, all while collecting coins and power-ups to assist you in your run.

- Pros:

- One of the most popular endless runner games
- Game is overall enjoyable and well received by players
 - Google Play Store 4.6 star rating (41 million reviews)
 - Apple App Store 4.7 star rating (400,600 reviews)
- Sense of competition with others is available through the online leaderboard that refreshes weekly
- Login system so all players can track their highscores individually and access their statistics at any time
- Prevents game from becoming monotonous through numerous methods:
 - Online leaderboard to compete with others across the world
 - Changing the theme of the game
 - Earning of in-game currency to purchase characters and boards in-game
 - Implementing challenges to earn in-game currency or exclusive characters
 - Challenging players to use in-game currency to purchase new items complete 'collections' of characters
 - Daily login rewards to encourage players to return to the game regularly
 - Daily & Seasonal quests for players to complete
 - Record of highscores & challenging users to beat their own highscores
 - Ability to upgrade power-ups to make them even more useful in-game

- Cons:

- Overly vibrant in-game colour scheme can be an eyesore to some users
- Excessive advertisements in-game ruins enjoyment for some users
- In-app purchases restrict users who don't play from having a better experience
- Menus on the game compared to the options within each menu item is confusing for users, menu labels are misleading
- Doesn't include any aspects where the player powers down by collecting negative power-ups

Evidence of In-Game Features

Challenging players to use in-game currency to purchase new items complete 'collections' of characters	Online leaderboard to compete with others across the world	Earning of in-game currency to purchase characters and boards in-game	Implementing challenges to earn in-game currency or exclusive characters	Daily login rewards to encourage players to return to the game regularly
				

Daily & Seasonal quests for players to complete	Doesn't include any aspects where the player powers down by collecting negative power-ups	Overly vibrant in-game colour scheme can be an eyesore to some users	In-app purchases restrict users who don't play from having a better experience
			

Features to Consider Applying to AstroSprint

- I will consider adding a login system to AstroSprint
 - This is so that different users can play AstroSprint and all of their data (e.g. in-game currency & high scores) will be stored separately
- Implementing challenges to earn in-game currency
 - I will consider adding this feature because I believe that it will be a driving force to encourage players to keep playing the game. This is because the endless runner aspect won't be the only possible option to play, so the game won't become monotonous, as the players have new things to explore, rather than the usual game mode.
- Clearly labelled menus with options within them being clear
 - This was a feature that Subway Surfers was missing, and I believe that having clear menus is a key to a viable and usable game, and will significantly improve the user interface experience, so I will consider adding this to AstroSprint
- Record of highscores & challenging users to beat their own highscores
 - I will consider adding this feature to AstroSprint to allow the users to be able to feel a sense of competition as they try to

beat their previous selves. The competition will drive the users to continue to play the game and excite them more.

- Negative power-ups

- I will consider including the use of negative power-ups within AstroSprint because I believe this is a good feature that increases the difficulty of the game and makes it more challenging for users. An overly easy game becomes extremely boring extremely quickly, so, apart from obstacles within the game, I will add negative power-ups to increase the difficulty

Geometry Dash Lite

This is a rhythm-based platformer game developed by RobTop Games. The aim of this game is to navigate through a music-based 2D world, while avoiding obstacles such as spikes, walls and creatures.

- Pros:

- Overall well received by players, with only general complaints about the escalation in difficulty
 - Google Play Store 4.3 star rating (7.53 million reviews)
 - Apple App Store 4.3 star rating (150.8 thousand reviews)
- Allows users to create and upload their own levels, preventing the game from becoming monotonous
- Starts with a clear menu that is simple, but understandable for users
- Mechanics & controls of game are easy to learn
- Achievements for the user to unlock to encourage them to keep playing & earn character customisation
- Practice mode, to enable players an easier way of practising playing before they play the main levels, with being able to respawn at any point
- Tutorial screen to show players basic controls of the game
- User-feedback mechanisms: When a menu item is being hovered over, it becomes larger

- Cons:

- Comes with a limited amount of pre-made levels (20), so the user must create their own levels, or play user generated ones to continue the entertainment
- Rhythm-based levels make the game significantly harder to play without audio
- Extreme escalation in difficulty from the start of the first level to the end of the first level
- Lots of character customisation hidden behind paywalls
- User-generated levels are inaccessible without paying

Features to Consider Applying to AstroSprint

- I will consider making sure AstroSprint also starts with a clearly labelled menu that is easier to understand

- This is because this is a feature that allows easy accessibility to the game. This ensures that, without any prior knowledge of the game, a new user is able to easily access the game
- I will consider making sure that the mechanics & controls of AstroSprint are easy to learn and clearly indicated
 - This is because it allows easy accessibility to playing the game. This ensures that, without any prior knowledge, a user can quickly and easily learn and enjoy playing the game
- I will consider making sure that the escalation in difficulty isn't as extreme as that of in Geometry Dash Lite
 - This is to prevent users from becoming easily frustrated with the game and quitting. This type of endless runner already requires fast reflexes and is unfamiliar to new users, so to allow users to ease into the game, I am going to make sure the escalation in difficulty is gradual and not extreme.
- I will consider not adding any paywalls to AstroSprint
 - This is because this greatly limits the potential for users to enjoy AstroSprint, and limits entertainment for those who do not have the money for in-app purchases or simply don't want to pay
- I am considering planning to make AstroSprint 2D, as there is already an overwhelming amount of 3D endless runner games
 - E.g. Subway Surfers, Minion Rush, Ratchet and Clank: Before the Nexus, the Hunger Games: Panem Run, Agent Dash, Sonic Dash, Panda Run, Running Fred, One Epic Knight, Aby Escape
- I will consider letting users have to earn certain character customisations
 - This is because it adds extra incentive to play the game, as decorating a base character or unlocking new characters can be a fun goal to work towards.
- I will consider adding user feedback mechanisms to show the user what they are currently hovering over
 - This is because hover effects make it easy for users to tell which parts of the interface they can interact with, helping them navigate more confidently. They also naturally guide users' attention to important actions, so they don't have to guess where to click.

Evidence of In-Game Features

Allows users to create and upload their own levels, preventing the game from becoming monotonous	Starts with a clear menu that is simple, but understandable for users	Comes with a limited amount of pre-made levels (20)
--	---	---



User Feedback
Mechanisms:
Enlarged hovered over items

Left (regular)
Right (Enlarged)



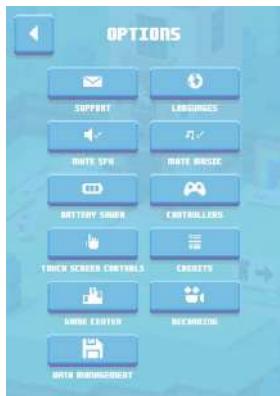
Crossy Road

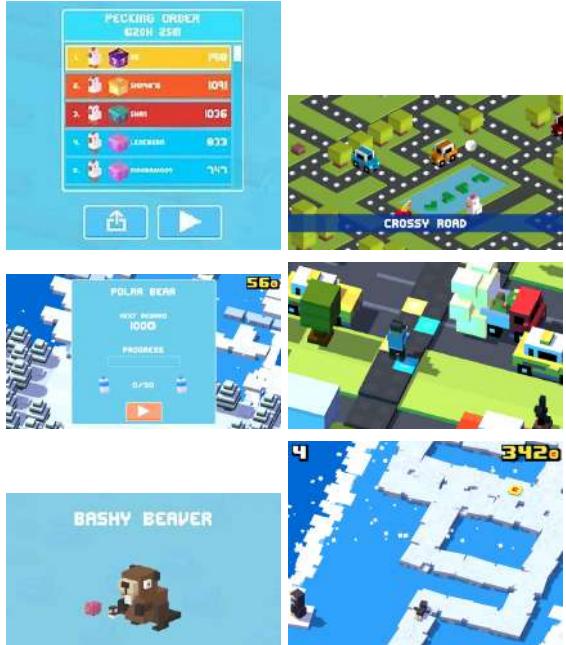
This is an 8-bit endless 'hopper' arcade game created by Hipster Whale. The aim of this game is to move a character through an endless path of static and moving obstacles as far as possible without dying.

- Scoring is based upon how far you get within the map
- In-game currency can be collected during gameplay
- Pros:
 - Overall well received by players
 - Google Play Store 4.5 star rating (4.55 million reviews)
 - Apple App Store 4.7 star rating (628.1 thousand reviews)
 - Game moves at the player's pace
 - When the player moves there is a certain amount of time that they have to move again before the game ends, but this timer resets every time the player makes a move forwards or backwards
 - Large catalogue of characters for players to choose from
 - With certain characters adding unique features to the game when played with (e.g. playing at night or different in-game music)
 - Backgrounds within game is strongly correlated to the available characters to play with, making the UI of the game very cohesive
 - Large variation of in-game obstacles (e.g. trains, rivers)
 - An option to conserve battery which reduces frame rate within the game
 - 6 different game modes to vary the experience of playing & to reduce monotony of the game
 - In-game lottery system to win prizes (coins, characters, etc.)
- Cons:
 - Game can be very slow paced in the beginning, making it seem monotonous in the beginning

- No negative power-ups to add challenge to the playing of the game
- No in-game tutorial when you first start playing the game

Evidence of In-Game Features

In-game currency can be collected during gameplay	Large catalogue of characters for players to choose from	Backgrounds within game is strongly correlated to the available characters to play with	Large variation of in-game obstacles	An option to conserve battery which reduces frame rate within the game
				

6 different game modes to vary the experience of playing & to reduce monotony of the game	In-game lottery system to win prizes (coins, characters, etc.)	Scoring is based upon how far you get within the map	Certain characters adding unique features to the game when played with
			

Features I Will Consider Adding To AstroSprint

- I will consider having a catalogue of characters that the player can choose from
 - This is because this is a feature that encourages users to keep playing the game in order to unlock all of the characters. This means that, even though the game is endless, it still has a 'goal' to complete
- I will consider adding in-game currency within AstroSprint that can be collected during gameplay
 - This is because it would be an incentive to encourage players to try and collect as much currency as possible, to be able to buy aspects of the game, such as characters, upgrades and varied environments
- I will also consider making sure that my background and characters are strongly correlated
 - This will make sure that the user interface of the game is cohesive, and will overall make the game more aesthetically pleasing to play, and just bring both the functionality and the design together
 - To develop on this point, I will also consider making sure that the power-ups also correlate, to increase cohesion
- I will consider adding a scoring system where scoring is based on how far you get in the map
 - This would enable me to have a linear scoring system that will be simple to explain to players, and will reduce the complexity of the game
- I will consider adding different game modes within the game
 - This will be good for reducing the monotony of the game, a limitation that would appear if there was only 1 way to play the game, with no levels.
- I will consider adding negative power-ups within AstroSprint
 - This is because it will be good for reducing the monotony of the game, and add a level of challenge that would make AstroSprint more appropriate for my stakeholders (13-18 year olds)
- I will consider adding an in-game tutorial to AstroSprint
 - This would be helpful for new users to the game so that they can learn how to play, and clearly understand the mechanics and controls of the game before they play, to make sure they can get the full experience, and not miss out because they do not understand the functionality of the game

1.3.2 - Stakeholder Questionnaire

I have done a questionnaire for my stakeholders to gauge their opinions on a sample of features I am considering adding to AstroSprint. In this form, a rating of 1 meant the feature will negatively impact the quality of AstroSprint, and 10 meant that the feature will positively impact the quality of AstroSprint. The respondents of this form were 15 people within the ages of 13-18

Feature & Rating	Response																						
<p>Implementing challenges to earn in-game currency 15 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr><td>1</td><td>0 (0%)</td></tr> <tr><td>2</td><td>0 (0%)</td></tr> <tr><td>3</td><td>0 (0%)</td></tr> <tr><td>4</td><td>0 (0%)</td></tr> <tr><td>5</td><td>0 (0%)</td></tr> <tr><td>6</td><td>1 (6.7%)</td></tr> <tr><td>7</td><td>1 (6.7%)</td></tr> <tr><td>8</td><td>2 (13.3%)</td></tr> <tr><td>9</td><td>6 (40%)</td></tr> <tr><td>10</td><td>5 (33.3%)</td></tr> </tbody> </table>	Rating	Count (%)	1	0 (0%)	2	0 (0%)	3	0 (0%)	4	0 (0%)	5	0 (0%)	6	1 (6.7%)	7	1 (6.7%)	8	2 (13.3%)	9	6 (40%)	10	5 (33.3%)	<p>Average Rating: 8.9/10 Due to the overall positive rating, I have decided to implement this feature into AstroSprint, as from the results in this form, it will improve the overall quality of AstroSprint</p>
Rating	Count (%)																						
1	0 (0%)																						
2	0 (0%)																						
3	0 (0%)																						
4	0 (0%)																						
5	0 (0%)																						
6	1 (6.7%)																						
7	1 (6.7%)																						
8	2 (13.3%)																						
9	6 (40%)																						
10	5 (33.3%)																						
<p>Clearly labelled menus, with options within them being clear 15 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr><td>1</td><td>0 (0%)</td></tr> <tr><td>2</td><td>0 (0%)</td></tr> <tr><td>3</td><td>0 (0%)</td></tr> <tr><td>4</td><td>0 (0%)</td></tr> <tr><td>5</td><td>0 (0%)</td></tr> <tr><td>6</td><td>0 (0%)</td></tr> <tr><td>7</td><td>0 (0%)</td></tr> <tr><td>8</td><td>0 (0%)</td></tr> <tr><td>9</td><td>6 (40%)</td></tr> <tr><td>10</td><td>9 (60%)</td></tr> </tbody> </table>	Rating	Count (%)	1	0 (0%)	2	0 (0%)	3	0 (0%)	4	0 (0%)	5	0 (0%)	6	0 (0%)	7	0 (0%)	8	0 (0%)	9	6 (40%)	10	9 (60%)	<p>Average Rating: 9.6/10 I believe that this is pivotal to a good experience in AstroSprint, and paired with the good reception from the form, I will implement this feature into AstroSprint.</p>
Rating	Count (%)																						
1	0 (0%)																						
2	0 (0%)																						
3	0 (0%)																						
4	0 (0%)																						
5	0 (0%)																						
6	0 (0%)																						
7	0 (0%)																						
8	0 (0%)																						
9	6 (40%)																						
10	9 (60%)																						
<p>Record of highscores & challenging users to beat their own highscores 15 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr><td>1</td><td>0 (0%)</td></tr> <tr><td>2</td><td>0 (0%)</td></tr> <tr><td>3</td><td>0 (0%)</td></tr> <tr><td>4</td><td>0 (0%)</td></tr> <tr><td>5</td><td>0 (0%)</td></tr> <tr><td>6</td><td>0 (0%)</td></tr> <tr><td>7</td><td>0 (0%)</td></tr> <tr><td>8</td><td>5 (33.3%)</td></tr> <tr><td>9</td><td>6 (40%)</td></tr> <tr><td>10</td><td>4 (26.7%)</td></tr> </tbody> </table>	Rating	Count (%)	1	0 (0%)	2	0 (0%)	3	0 (0%)	4	0 (0%)	5	0 (0%)	6	0 (0%)	7	0 (0%)	8	5 (33.3%)	9	6 (40%)	10	4 (26.7%)	<p>Average Rating: 8.9/10 Due to these results, I will implement into AstroSprint a display of highscores & similarly to 'Subway Surfers', I will implement this as a feature.</p>
Rating	Count (%)																						
1	0 (0%)																						
2	0 (0%)																						
3	0 (0%)																						
4	0 (0%)																						
5	0 (0%)																						
6	0 (0%)																						
7	0 (0%)																						
8	5 (33.3%)																						
9	6 (40%)																						
10	4 (26.7%)																						
<p>Negative power-ups 15 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr><td>1</td><td>1 (6.7%)</td></tr> <tr><td>2</td><td>0 (0%)</td></tr> <tr><td>3</td><td>0 (0%)</td></tr> <tr><td>4</td><td>1 (6.7%)</td></tr> <tr><td>5</td><td>1 (6.7%)</td></tr> <tr><td>6</td><td>3 (20%)</td></tr> <tr><td>7</td><td>3 (20%)</td></tr> <tr><td>8</td><td>2 (13.3%)</td></tr> <tr><td>9</td><td>3 (20%)</td></tr> <tr><td>10</td><td>1 (6.7%)</td></tr> </tbody> </table>	Rating	Count (%)	1	1 (6.7%)	2	0 (0%)	3	0 (0%)	4	1 (6.7%)	5	1 (6.7%)	6	3 (20%)	7	3 (20%)	8	2 (13.3%)	9	3 (20%)	10	1 (6.7%)	<p>Average Rating: 6.8/10 Due to the spread of results for this feature, I think I will have more than 1 mode within AstroSprint, so that some are harder than others. I think there could be a hard mode with negative power-ups and harder obstacles</p>
Rating	Count (%)																						
1	1 (6.7%)																						
2	0 (0%)																						
3	0 (0%)																						
4	1 (6.7%)																						
5	1 (6.7%)																						
6	3 (20%)																						
7	3 (20%)																						
8	2 (13.3%)																						
9	3 (20%)																						
10	1 (6.7%)																						
<p>Mechanics & controls of my game are easy to learn and clearly indicated 15 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr><td>1</td><td>0 (0%)</td></tr> <tr><td>2</td><td>0 (0%)</td></tr> <tr><td>3</td><td>0 (0%)</td></tr> <tr><td>4</td><td>0 (0%)</td></tr> <tr><td>5</td><td>0 (0%)</td></tr> <tr><td>6</td><td>0 (0%)</td></tr> <tr><td>7</td><td>0 (0%)</td></tr> <tr><td>8</td><td>2 (13.3%)</td></tr> <tr><td>9</td><td>6 (40%)</td></tr> <tr><td>10</td><td>7 (46.7%)</td></tr> </tbody> </table>	Rating	Count (%)	1	0 (0%)	2	0 (0%)	3	0 (0%)	4	0 (0%)	5	0 (0%)	6	0 (0%)	7	0 (0%)	8	2 (13.3%)	9	6 (40%)	10	7 (46.7%)	<p>Average Rating: 9.3/10 Due to the overall extremely positive response to this feature, I will make sure that the mechanics & controls of AstroSprint are easy to learn and clearly indicated</p>
Rating	Count (%)																						
1	0 (0%)																						
2	0 (0%)																						
3	0 (0%)																						
4	0 (0%)																						
5	0 (0%)																						
6	0 (0%)																						
7	0 (0%)																						
8	2 (13.3%)																						
9	6 (40%)																						
10	7 (46.7%)																						

<p>Escalation in difficulty of the game isn't extreme 15 responses</p> <table border="1"> <thead> <tr> <th>Difficulty Level</th> <th>Count</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>0</td><td>(0%)</td></tr> <tr><td>2</td><td>1</td><td>(6.7%)</td></tr> <tr><td>3</td><td>0</td><td>(0%)</td></tr> <tr><td>4</td><td>1</td><td>(6.7%)</td></tr> <tr><td>5</td><td>2</td><td>(13.3%)</td></tr> <tr><td>6</td><td>2</td><td>(13.3%)</td></tr> <tr><td>7</td><td>3</td><td>(20%)</td></tr> <tr><td>8</td><td>2</td><td>(13.3%)</td></tr> <tr><td>9</td><td>3</td><td>(20%)</td></tr> <tr><td>10</td><td>1</td><td>(6.7%)</td></tr> </tbody> </table>	Difficulty Level	Count	Percentage	1	0	(0%)	2	1	(6.7%)	3	0	(0%)	4	1	(6.7%)	5	2	(13.3%)	6	2	(13.3%)	7	3	(20%)	8	2	(13.3%)	9	3	(20%)	10	1	(6.7%)	<p>Average Rating: 6.8/10 Due to large spread of results on this feature, I have decided to make the escalation of difficulty only a feature in the harder mode of the game, just to give users a choice of how much they want the escalation of difficulty to be</p>
Difficulty Level	Count	Percentage																																
1	0	(0%)																																
2	1	(6.7%)																																
3	0	(0%)																																
4	1	(6.7%)																																
5	2	(13.3%)																																
6	2	(13.3%)																																
7	3	(20%)																																
8	2	(13.3%)																																
9	3	(20%)																																
10	1	(6.7%)																																
<p>Make my game 2D, as there is already an overwhelming amount of 3D endless runner games 15 responses</p> <table border="1"> <thead> <tr> <th>Difficulty Level</th> <th>Count</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>0</td><td>(0%)</td></tr> <tr><td>2</td><td>0</td><td>(0%)</td></tr> <tr><td>3</td><td>0</td><td>(0%)</td></tr> <tr><td>4</td><td>0</td><td>(0%)</td></tr> <tr><td>5</td><td>1</td><td>(6.7%)</td></tr> <tr><td>6</td><td>2</td><td>(13.3%)</td></tr> <tr><td>7</td><td>1</td><td>(6.7%)</td></tr> <tr><td>8</td><td>3</td><td>(20%)</td></tr> <tr><td>9</td><td>4</td><td>(26.7%)</td></tr> <tr><td>10</td><td>4</td><td>(26.7%)</td></tr> </tbody> </table>	Difficulty Level	Count	Percentage	1	0	(0%)	2	0	(0%)	3	0	(0%)	4	0	(0%)	5	1	(6.7%)	6	2	(13.3%)	7	1	(6.7%)	8	3	(20%)	9	4	(26.7%)	10	4	(26.7%)	<p>Average Rating: 8.3/10 I think that I will make AstroSprint 2D because it will also make the creation of the game less complex, and from the form, it is clear that it will improve the game quality</p>
Difficulty Level	Count	Percentage																																
1	0	(0%)																																
2	0	(0%)																																
3	0	(0%)																																
4	0	(0%)																																
5	1	(6.7%)																																
6	2	(13.3%)																																
7	1	(6.7%)																																
8	3	(20%)																																
9	4	(26.7%)																																
10	4	(26.7%)																																
<p>No paywalls 15 responses</p> <table border="1"> <thead> <tr> <th>Difficulty Level</th> <th>Count</th> <th>Percentage</th> </tr> </thead> <tbody> <tr><td>1</td><td>0</td><td>(0%)</td></tr> <tr><td>2</td><td>0</td><td>(0%)</td></tr> <tr><td>3</td><td>0</td><td>(0%)</td></tr> <tr><td>4</td><td>0</td><td>(0%)</td></tr> <tr><td>5</td><td>1</td><td>(6.7%)</td></tr> <tr><td>6</td><td>1</td><td>(6.7%)</td></tr> <tr><td>7</td><td>1</td><td>(6.7%)</td></tr> <tr><td>8</td><td>3</td><td>(20%)</td></tr> <tr><td>9</td><td>4</td><td>(26.7%)</td></tr> <tr><td>10</td><td>5</td><td>(33.3%)</td></tr> </tbody> </table>	Difficulty Level	Count	Percentage	1	0	(0%)	2	0	(0%)	3	0	(0%)	4	0	(0%)	5	1	(6.7%)	6	1	(6.7%)	7	1	(6.7%)	8	3	(20%)	9	4	(26.7%)	10	5	(33.3%)	<p>Average Rating: 8.5/10 I will definitely have no paywalls within AstroSprint, and this form has confirmed how having no paywalls significantly improves the gaming experience.</p>
Difficulty Level	Count	Percentage																																
1	0	(0%)																																
2	0	(0%)																																
3	0	(0%)																																
4	0	(0%)																																
5	1	(6.7%)																																
6	1	(6.7%)																																
7	1	(6.7%)																																
8	3	(20%)																																
9	4	(26.7%)																																
10	5	(33.3%)																																

1.4 - Hardware & Software Requirements

1.4.1 - Developer Requirements

Hardware Requirements

- The development of AstroSprint will not require any specialist equipment, only a standard laptop with an operating system that can run the Unity game engine, such as macOS, Microsoft Windows or Linux.
- I will be using the built in programming language of the Unity game engine, which is C# (c-sharp).

Software Requirements

- The development of AstroSprint will require the game engine Unity, to support the creation of the game and the environment it will run in. The software Visual Studio code will also be used in the development of AstroSprint, to write the scripts for each game object within AstroSprint.
- To implement the database within AstroSprint, I will be using Microsoft Playfab, as outlined in [2.8.5 - Database Implementation](#)

Unity - A cross-platform game engine developed by Unity Technologies

- Released in June 2005 at Apple Worldwide Developers Conference as a Mac OS X game engine
- It is able to create both 2D & 3D games, on mobile, computer & VR
 - In 2D: Allows importation of sprites & an advanced 2D renderer
 - In 3D: Allows specification of texture compression, mipmaps, resolution settings for each platform that the game engine supports,
 - As well as support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects
- After creating the game, I will export it as a executable file, in which the user can access
- Sources: [Unity](#), [Unity Education](#)

Microsoft PlayFab - a complete backend platform for live games with managed game services, real-time analytics, and LiveOps

- Acquired by Microsoft in 2018
- Hosts databases, services, authentication, and integration for a variety of games
- As it is compatible with Unity, I will be using it as a real-time database that can hold detailed information about AstroSprint
- Sources: [Azure PlayFab](#)

1.4.2 - User Requirements

Hardware Requirements

- The playing of AstroSprint will not require any specialist equipment, only a standard laptop with an operating system that runs on macOS, Microsoft Windows or Linux.

Software Requirements

- The playing of AstroSprint will not require any specialist software.

1.5 - Success Criteria

1.5.1 - Menus & UI Criteria

Success Criteria	Overall Criteria No.	Justification	How is it Measurable?
All menus titled	1	Titles are needed so that the user knows what menu they are on, and to summarise the function of that menu	Go through of all menus with the title shown

Clearly labelled menus, with options within them being clear	2	Key to a viable and usable game, and will significantly improve the user interface experience, by making the game more understandable & accessible	Go through the full game, showing all of the menu titles and the options within them, like buttons leading to clear options
Main Menu UI	3	Main Menu is the starting point of the game. It is the first thing that the user sees when they open the game, and it facilitates player login or sign up account creation for personalised experiences	Video of main menu and all interactable features
Login Menu UI	4	Necessary to the game for users to be able to log into their account and keep all of their previous data (e.g. coin balance or highscores). Login menu allows users to access their accounts and their data.	Video of login menu and all interactable features
Sign Up Menu UI	5	Necessary for the game in order for users to create accounts and save their data, which can be used later (e.g. coins collected being saved to external database so that	Video of sign up menu and all interactable features

		later on, they can use it to make in-game purchases), as well as necessary to allow new users to play my game.	
Play Menu UI	6	Necessary to enable users to see all the options to interact with the game after signing in. The play menu allows users to access the gameplay and other customisation menus	Video of play menu and all interactable features
Tutorial Screen UI	7	Necessary to show users the controls of the game	Video of tutorial screen and all interactable features
Character Menu UI	8	Allows users to be able to change their in-game avatar	Video of character menu and all interactable features
Abilities Menu UI	9	Allows users to be able to see what the power ups available are, and to upgrade them.	Video of abilities menu and all interactable features
Pause Menu UI	10	Allows users to pause in-game	Video of pause menu and all interactable features
Game Summary UI	11	Allows users to see their statistics from the game they just played after they have died	Video of game summary menu and all interactable features
Back buttons on all menus	12	This is essential for navigation in the menus, so when a user is done interacting	Video showing use of back buttons on all menus

		with a specific menu, they can go back and do something else	
Non-bright, cohesive colour scheme	13	This is to ensure that the colour scheme is palatable for all audiences.	Primary user shown video of gameplay and review
Varied environments, with users being able to play in places with different backgrounds	14	This is to ensure there is variation in the game, as there are no levels	Video of user playing in all of the different environments
Login system	15	Necessary to the game for users to be able to log into their account and keep all of their previous data (e.g. coin balance or highscores). Login menu allows users to access their accounts and their data.	Video of login menu being used, and a pre-existing user being able to log in.
Sign up system	16	Necessary for the game in order for users to create accounts and save their data, which can be used later (e.g. coins collected being saved to external database so that later on, they can use it to make in-game purchases), as well as necessary to allow new users to play my game.	Video of sign up menu being used and new user being able to sign up
Catalogue of characters that the player can choose from	17	This enables the user to have variety of characters to	Video of character menu being used, showing all options for characters within

		play with, to enhance the gaming experience	them
Input validation for email when signing up	18	This is to ensure that all data going into the database is in a valid email format, and to make sure that when signing up, all details are present (validation includes a presence check for the email input field, and a format check)	Video of invalid and valid inputs being input into the email field and corresponding error messages
Input validation for username when signing up	19	Necessary to ensure that the user has entered something into the username field, and if they have, that it is in the correct format	Video of invalid and valid inputs being input into the username field and corresponding error messages
Input validation for password when signing up	20	Necessary to ensure that the user has entered something into the username password, and if they have, that it is between 8 and 16 characters long	Video of invalid and valid inputs being input into the password field and corresponding error messages
Comparison between password and confirm password fields	21	Necessary to ensure that the user correctly remembers their password, and doesn't have to reset it, as the game will have no reset password function	Video of passwords that match and don't match, and corresponding error messages for passwords that don't match
Coins balance retrieved from external database in	22	Necessary so that the user knows how many coins	Video of coin balance being shown on-screen from a pre-existing

abilities menu		they have when they are trying to purchase abilities	user that just logged in
Coin balance updated in external database after every upgrade purchase	23	This is so that on the next opening of the menu, the correct coin balance is shown	Video of user upgrading power ups using coins and screenshot of the coin balance in external database after these purchases have been made
Indication of which character was chosen in Character Menu	24	Necessary so that the user knows what character they have selected	Video of a character being selected and its corresponding indicator showing an image of that character.
Indicator of each power up's level	25	Necessary so that the user knows how much their power up has been upgraded	Video of user opening the abilities menu, and being able to see, on inspection, see something that shows the power up's level
There is a default selected character	26	Necessary so that, if the player doesn't use the character menu before playing, they have a sprite to play as	Video of a new user opening the character menu and there being an indicator showing their currently selected character, before they have selected anything themselves.

1.5.2 - Gameplay Criteria

Success Criteria	Overall Criteria No.	Justification	How is it Measurable?
Implementing challenges to gain in-game currency	27	Necessary to ensure that the game does not become boring, and the user always have something to work towards	Video of challenges menu showing all of the different challenges, how much currency they are worth on completion, and the current progress towards completing the challenge

Game being 2D	28	There is already an overwhelming amount of 3D endless runner games, so 2D would be something that revolutionises the game market, just as I had described wanting to do	Video of every single menu and all of the gameplay and inspection on what planes the game exists on
Slow escalation in difficulty in the easier mode of the game	29	Necessary to ensure that users have the option to have an easier playing experience, which is more relaxing and fulfilling to play	Video of easy mode gameplay, showing the horizontal movement getting progressively faster
Indication of what power ups are active to user	30	Necessary so the player knows the abilities available to them, as the controls will differ depending on the power up active	Video of gameplay, where there is a collision with a power up icon, with text shown at the top of the screen, naming the power up.
Indication of power up remaining duration when active	31	Necessary so the player knows how long the abilities will be available to them, by knowing how long they have left	Video of gameplay, where there is a collision with a power up icon, with a visual indicator showing the duration of the power up.
Power-ups	32	Necessary to create a more varied gaming experience, so the game doesn't become as monotonous	Video of gameplay, showing the use of power ups
Power up duration according to level upgraded	33	Necessary to ensure that there is an incentive to upgrade power ups, and the levels are	Video of upgrading the power ups on the abilities menu, and then going to the gameplay and timing how long it takes for

		utilised for something	a power ups effects to stop working
Scoring system based on how long you last in-game	34	Necessary to create a simplistic game scoring system that is easy for players to understand	Video of gameplay, where scores are shown at the top right hand corner of the screen, and by inspection, seeing if the score increases as time passes, in a regular pattern.
Different game modes	35	Necessary to enable players to change the experience they have within the game, and allows players, especially new ones, to gain skills in the game without immediately doing a mode that is too hard	Video of gameplay where the user clicks on 2 different buttons to lead to 2 different game modes, that are clearly distinguishable
Implementation of negative power ups in a harder mode	36	Necessary to ensure that users get to have something that makes their gaming experience more difficult and challenging, encouraging them to play more	Video of gameplay, where a player collides with a negative power up, and gets the corresponding negative effect for 30 seconds.
Implementation of a sharp escalation in difficulty in the harder mode, in comparison to the easy mode	37	Necessary to ensure that there is a strong escalation in difficulty	Video of gameplay of easy mode next to a video of gameplay of hard mode, showing how the speed increase in hard mode is quicker than that of the easy mode.
Game is endless	38	Necessary so that the game has clearer visuals and is easier to control for users, so that they can quickly understand the	Video of gameplay, and on inspection, the foreground should be looping continuously through the same few options

		gameplay and remain engaged	
Random generation of in-game currency within the playing environment	39	Necessary so that the player has an incentive (collecting in-game currency) to keep the game engaging. The random generation is also necessary so that the game doesn't become too predictable	Video of gameplay, and inspecting the generation of coins to see if they are generated at both random time intervals and at random positions on-screen
Collection of in-game currency after colliding with it	40	Necessary so that the player is actually able to collect in-game currency	Video of gameplay, where the player collides with a coin in-game and collects them, as shown in a text element on-screen
Ability to upgrade power-ups outside of the playing environment	41	Necessary so that the player has something to work towards using their in-game currency, so that the game remains engaging.	Video of abilities menu and of user using coins to upgrade the power ups
Ability to pause in-game	42	Necessary so that the player can quickly leave the game, or momentarily stop playing for any reason and return to the gameplay without losing previous progress.	Video of gameplay, where the user clicks the pause button and is then taken to the pause menu
Maintain current score after coming out of pause menu	43	Necessary so that pausing the game doesn't cause any disadvantage to the player	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same score as they had before they paused

Maintains player's state (alive/dead) after pause	44	Necessary so that pausing the game doesn't cause any disadvantage to the player	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same state as they had before they paused
Maintains current coins collected after pause	45	Necessary so that pausing the game doesn't cause any disadvantage to the player	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same coins as they had before they paused
Maintains power up remaining duration after pause	46	Necessary so that pausing the game doesn't cause any disadvantage to the player	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same power up duration remaining as they had before they paused
Player death when player leaves the view	47	Necessary so that the game, which is currently endless, has a stopping condition.	Video of gameplay where the player leaves the view of the game and them being taken to the game summary menu after
Player death after collision with obstacle	48	Necessary so that the game, which is currently endless, has a stopping condition. Also necessary so that the player has something to actively try to avoid	Video of gameplay where the player collides with an obstacle and then is taken to the game summary menu after
Random generation of power ups	49	Necessary so that the player has a temporary boost in-game (collecting a power up) to keep the game engaging. The random generation is also necessary	Video of gameplay, and inspecting the generation of individual power ups to see if they are generated at both random time intervals and at random positions on-screen

		so that the game doesn't become too predictable, in which power up will spawn next.	
After player death, in-game statistics shown in the game summary	50	Necessary so that the player knows the results of their previous game, and can possibly see if they have set a new highscore	Video of gameplay, where the player dies and is taken to the game summary menu, where accurate statistics from the previous game are shown in on that menu.
After player death, in-game statistics transferred to external database	51	Necessary so that the coin balance can be updated every time the user completes a game so the user can keep working towards increasing their coin balance. Also necessary for seeing when a new high score is set, as it needs to compare the current score to the previous high score and if it is larger than the current highscore, alert the user.	Video of the abilities menu, where the player views the coin balance, then goes into the gameplay, collects coins, where the player dies and is taken to the game summary menu, exits and returns to the Play Menu, then new value of coin balance is checked in the external database to be increased from the original value.

1.5.3 - External Database Criteria

Success Criteria	Criteria No.	Justification	How is it Measurable?
Stores highscores	52	Necessary so that players can have a sense of challenge in trying to beat themselves, even after logging	Screenshot of external database for a user showing the field that contains their highscore

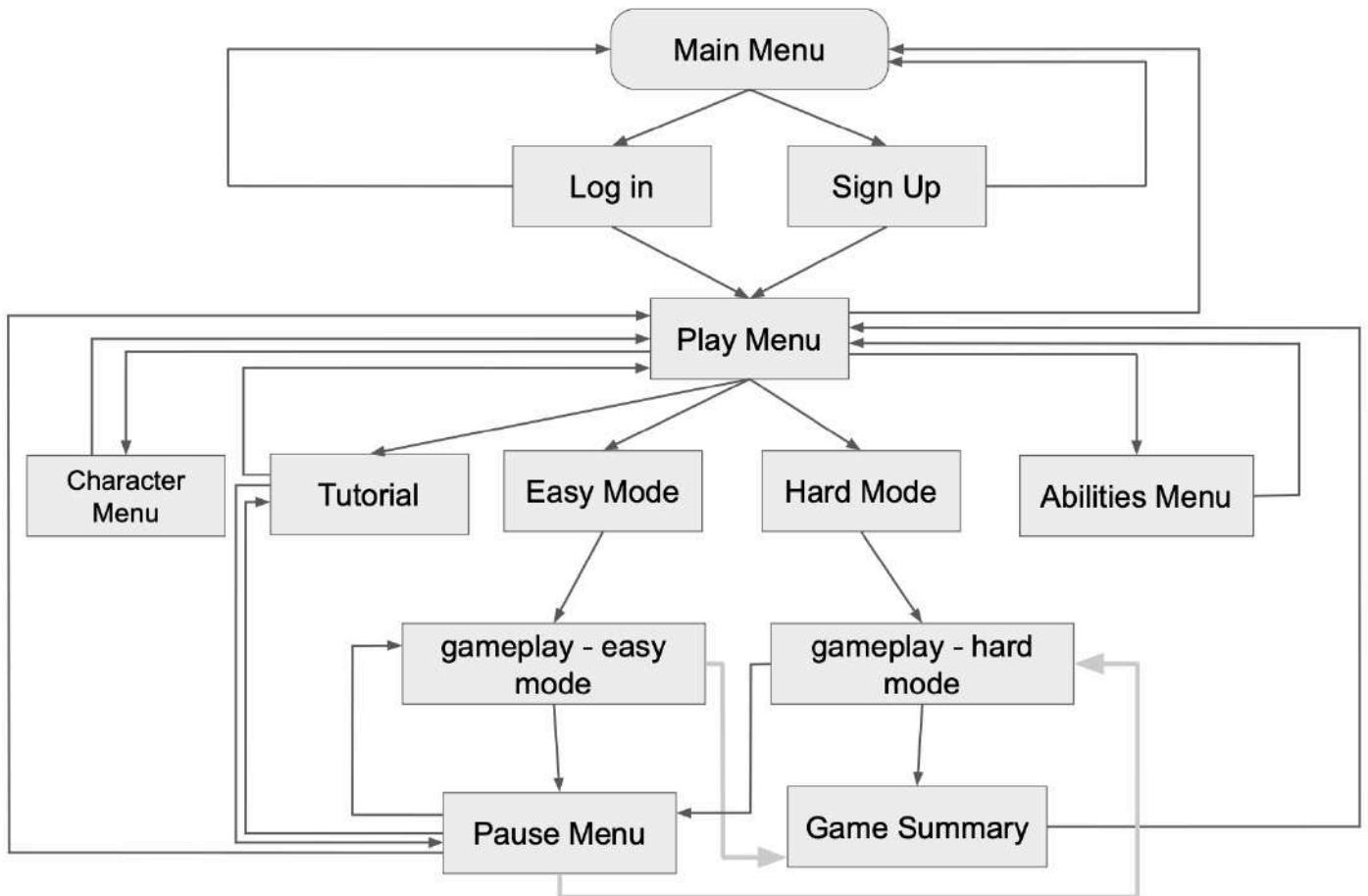
		out, as storing data in external database ensure that the highscore data is not lost	
Stores coin balance	53	Necessary to ensure that coin balance is maintained even when the player is not logged in. Beneficial to players as it means that they can build up their coin balance to make purchases and have this progress maintained when they log back in	Screenshot of external database for a user showing the field that contains coin balance
Stores selected character	54	Necessary so that the player can keep playing as the same character even after logging out and logging back in. Beneficial to the player as it means that they don't have to select their preferred each time they log in, and go straight to the gameplay.	Screenshot of external database for a user showing the field that contains their selected character
Stores usernames from sign up	55	Necessary so that data can be stored to a specific user, and usernames are memorable, so users can remember them easily and remember their username and log back in to access their data.	Screenshot of external database for a user showing the field that contains their username

Stores passwords from sign up	56	Necessary so that the user can log in and access their data after signing up. This means that the player can retain their game data and access it when needed.	Screenshot of external database for a user showing the field that contains their password
Stores emails from sign up	57	Necessary for sign up as they ensure security & allow communication. In future, developers of the game can send game updates or security alerts	Screenshot of external database for a user showing the field that contains their email
Stores data from 1 user all in the same place	58	Necessary so that all data is easily accessible for each user. Keeping user data in separate tables for each user ensures that data is never mixed up when retrieving.	Screenshot of the external database showing all of the corresponding fields for all of the user data in one place
Storage of power up levels for each player	59	Necessary so players can retain their upgrades even when they have logged out of the game.	Screenshot of external database for a user showing the field that contains the power up levels
Default power up level 1 (Duration 30s)	60	Necessary so that the player has something to work towards (purchasing more upgrades for power ups), and means all players start on the same level, ensuring fairness in the game.	Screenshot of external database for new user showing that the default power up levels are all 1
Default coin balance	61	Necessary so that	Screenshot of

of 0		the player has something to work towards (gaining more coins), and means all players start on the same level, ensuring fairness in the game.	external database for new user showing that the default coin balance is 0
Default highscore of 0	62	Necessary so that the player has something to work towards (beating their own highscore), and means all players start on the same level, ensuring fairness in the game.	Screenshot of external database for new user showing that the default highscore is 0

2 - Design

2.1 - Menus Flowchart



This is a flowchart that describes the menu that the user will be led to after choosing the options outlined in the main menu.

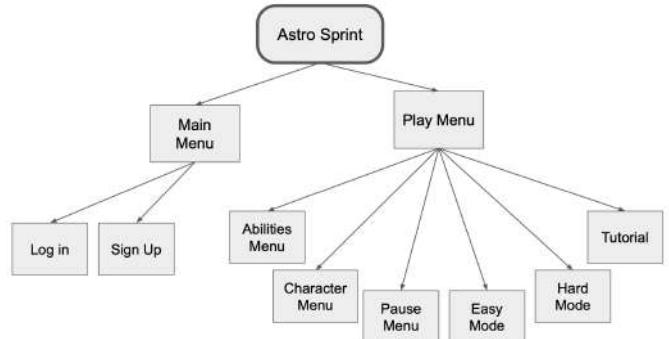
- As outlined in the previous section, there will be 2 options in the main menu, login & sign up
- After logging in or signing up, this leads to the play menu where you can then select either easy mode, hard mode, tutorial or the abilities menu
 - If the user wishes to return to the main menu, they can.
 - If choosing a game mode, the user is taken to the corresponding game mode that they choose
 - While in a game mode, the user can choose to pause, and the pause menu will enable them to either go to the play menu or return to the game mode they were playing previously
 - After a player death, the user is taken from their game mode to a game summary, and then back to the play menu

2.2 - Game Decomposition

2.2.1 - Main Decomposition Diagram

Justification

I have split AstroSprint into 2 main subproblems (Main Menu & Play Menu), which are separated into 2 and 6 subproblems respectively. I have done this so that I can break down the development of my project into separate modules that I can solve individually, and then bring them together to have my final game.

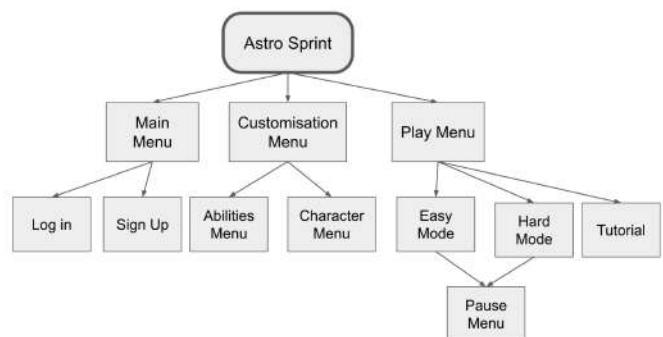


I am going to tackle the main menu first, starting with the sign in, then log up. I am doing the main menu first because this is the forefront of AstroSprint, and will be where the user starts off. I will then do the play menu, and tackle it in this order: tutorial, easy mode, pause menu, abilities menu, character menu and finally hard mode. I will bring them together through the various buttons on the screen being used as inputs, so when pressed they are connected to their corresponding screen.

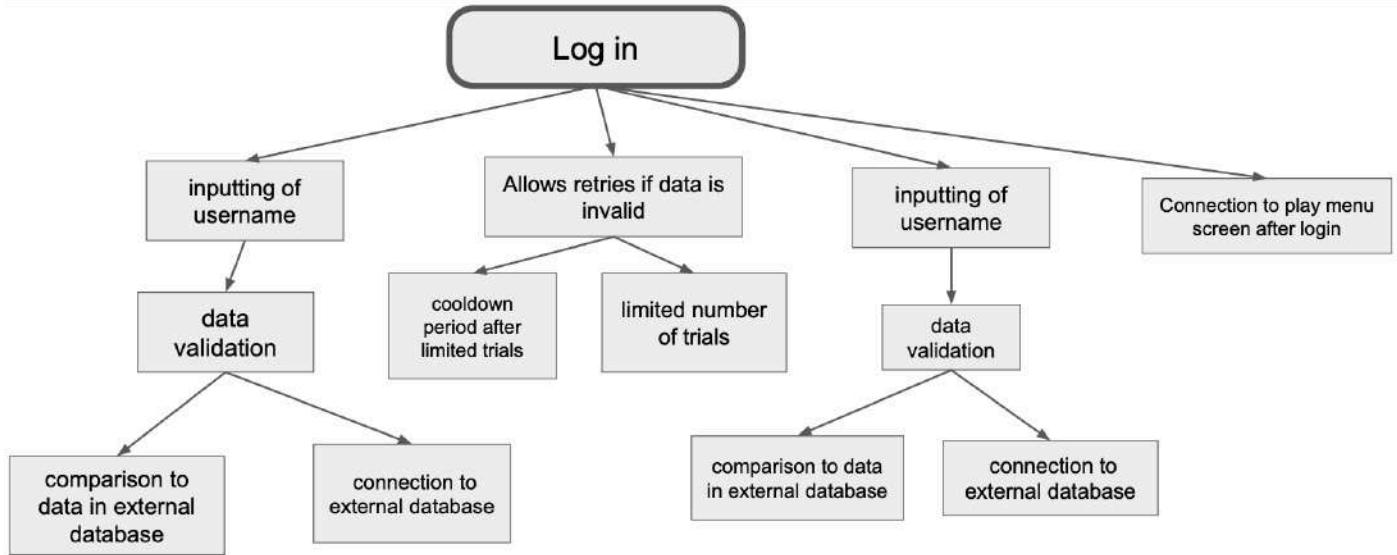
The programming constructs I will be using include selection, which I will be using for when the player decides to select which screen they want to go onto first. I will also be using sequence in order to make my program begin at the main menu before going to any other sections. These divisions will make development easier as I believe it is more simple to code for each section of the game individually, rather than trying to do it all at once. I also believe that it will be easier to develop them separately and bring them together because it reduces the complexity of the program.

2.2.1.1 - Alternative Decomposition

This is an alternative decomposition of my game, AstroSprint. In this decomposition, there are 3 big menus, the main menu, the customisation menu & the play menu, each with their own menus within them. Although I could structure the game like this, it would make the navigation of the game much more tedious, as the main menu would lead to the play menu, in which the user would have to go to the customisation menu to finally get to the abilities menu or character menu. The fewer large menus there are, the easier the game is to navigate. Therefore, I will not be structuring the game in this way.



2.2.2 - Login System Decomposition



Justification

I have separated the login section of the program into 4 main components, which are split into 3, 2, 3 & 0 respectively, as shown in the diagram above. I have separated them to ensure that I tackle each problem separately in their entirety. I will do this by looking at each sub-problem, and from the bottom up, create a flowchart for its module, and plan out the variables & processes associated with them.

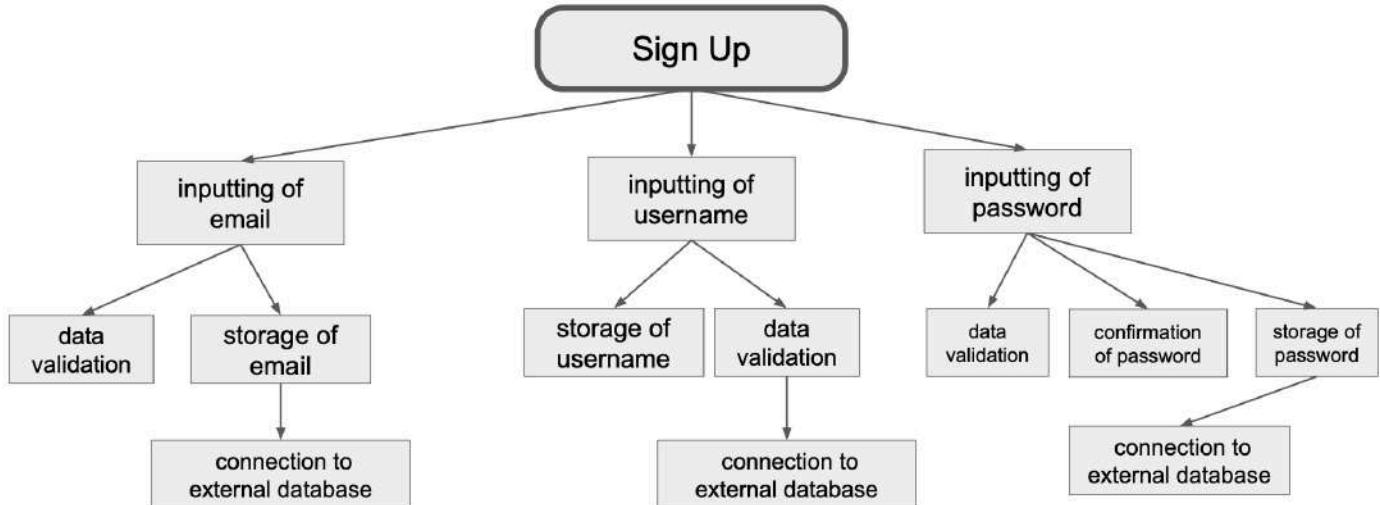
I am going to design & develop these starting from the leftmost bottom one in the diagram (comparison to external database) then moving across to the right and then going up, until I have developed the entire login program. After developing all of the individual modules, I will bring them together to create the final program.

I am developing in this order because the items lower down in the diagram are the foundations for creating the aspects higher up in the diagram. The higher aspects do not exist without the lower aspects, so I have to develop the lower aspects first to be able to create the higher up aspects.

I will connect all of these aspects together through variables that will be used across all of the separate sections to connect their data together, as well as values from the lower down aspects being passed into other sections to connect their function

I am going to use the programming constructs of sequence, selection & iteration. I will use sequence in many ways, such as to make the program start from comparing the data to that of the data in the database, then seeing if it matches, then returning whether the data is valid or not.

2.2.3 - Sign Up System



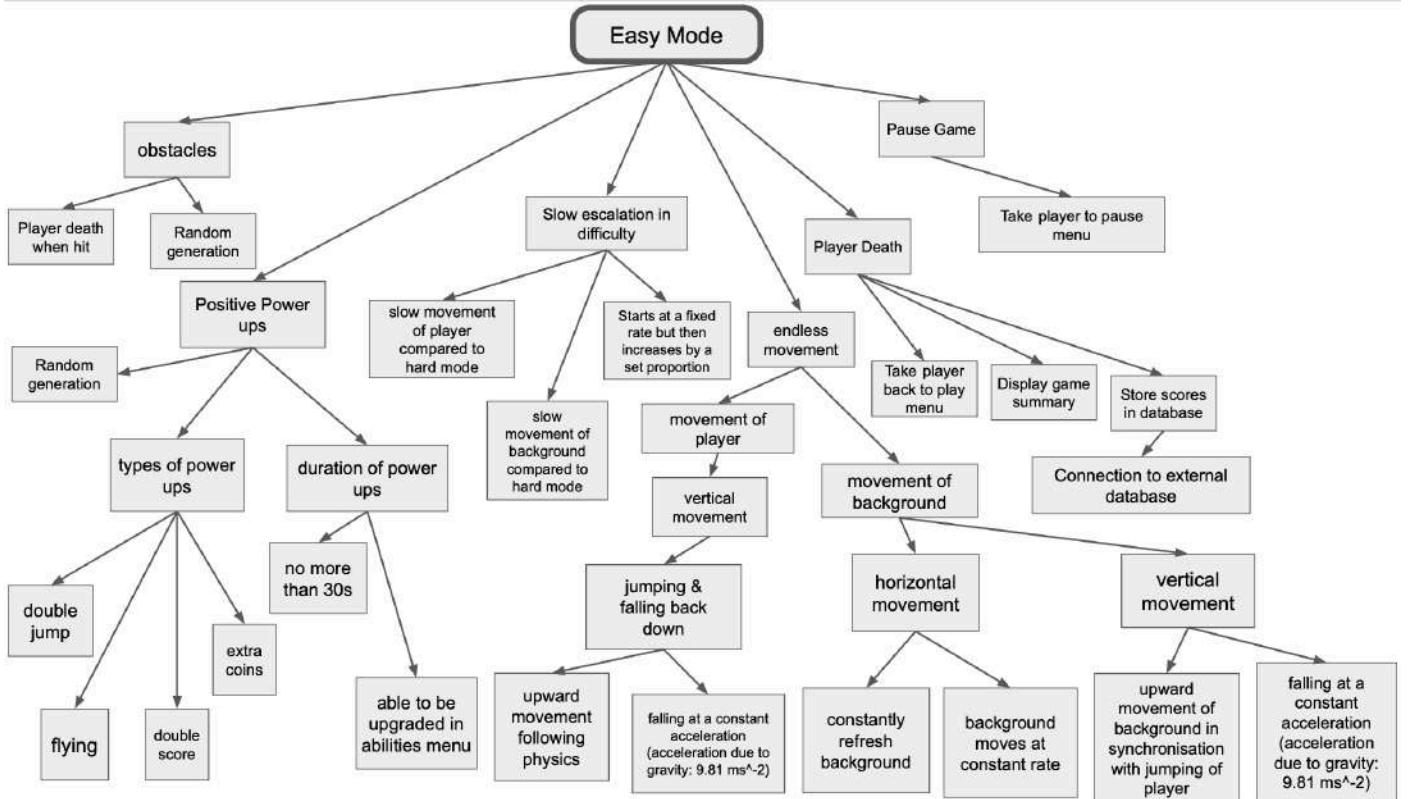
I have separated the idea of a sign up into 3 sub-sections, which have 3, 3 and 4 sub-sub-sections respectively. I have split it up in this way to promote a bottom-up approach to problem solving & have the smaller foundations build up to a final product, rather than it all having to be tackled in 1 go. This promotes modularisation & will also allow me to be able to reuse parts of the solution to help inform other aspects of it, overall making development efficient

I am going to tackle the input of username first, then the inputting of email, then inputting of password. I was going to do it in this order as I believe that the code for username would have some reusable components that could be utilised within the code for creating an email. I believe that the inputting of password has the least in common with both of these, therefore, I will be coding it last

I will bring them together through the utilisation of the submission button, which will check all if these fields simultaneously for the various variable checks, outlined in [2.7.1 - User Sign Up Details](#)

I will be using the programming constructs of sequence, selection and iteration to be able to implement this.

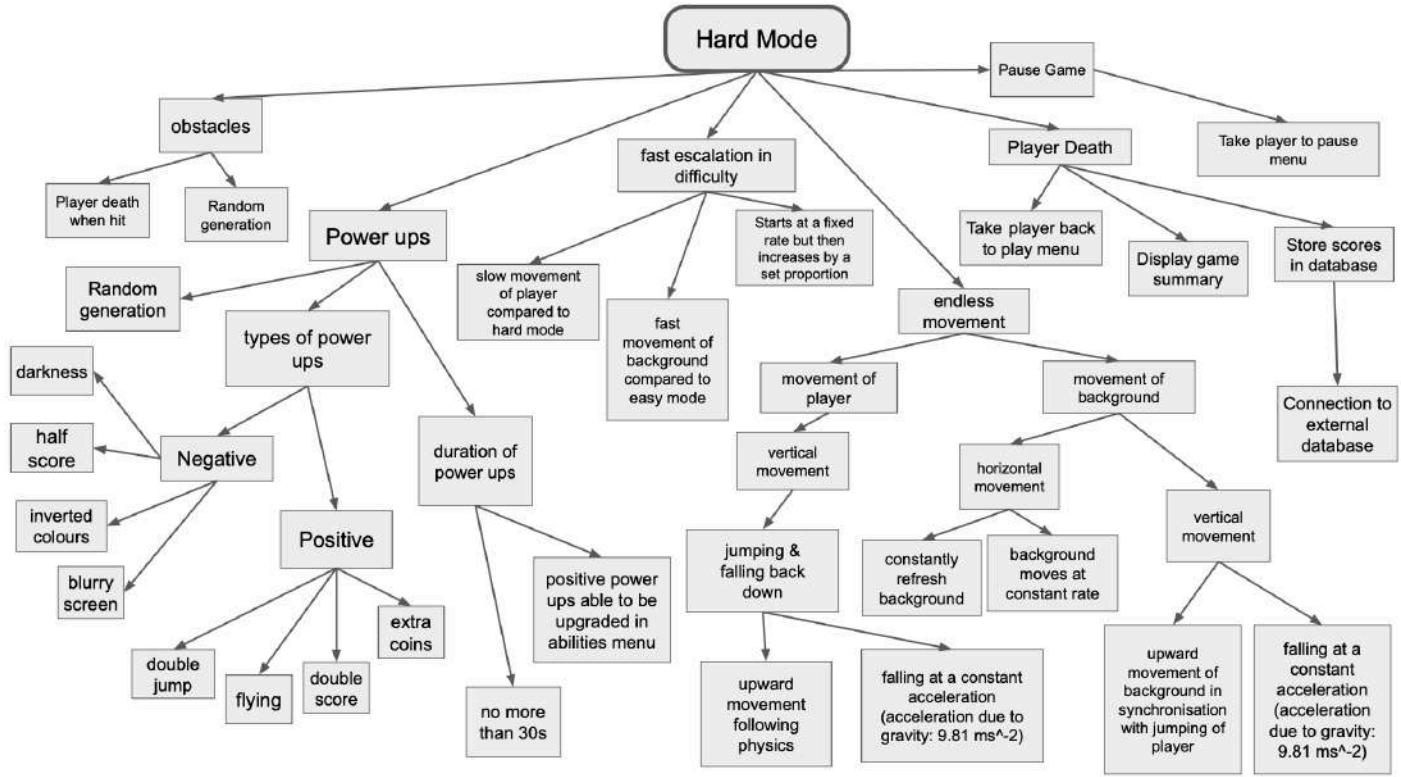
2.2.4 - Easy Mode Decomposition



I have separated the easy mode into 4 sections, each with a multitude of sub-sections within them all to be able to break down the problem as simply as possible. I believe that making an entire game mode is a complex process, and so by splitting it up, I can view it as the conquering of smaller challenges which lead to the creation of the final solution. I have especially split up the power ups so that they can be reusable for the other game mode, hard mode.

I am going to tackle the endless movement first, then the power ups, then the slow escalation in difficulty, then obstacles, then player death and lastly, the pause game. I am purposefully doing the endless movement first because it is the environment that the other 2 aspects exist in. I am doing the power ups next, as I believe that it will be a good benchmark to test the game, and that aspect will need to be altered when I do the slow escalation in difficulty, which will come last as it just brings the game together. I am then doing obstacles because that is what causes player death. I am doing the pause game last, as that is intertwined with the development of the pause menu. I will also bring all of these separate aspects together through implementing them within the same scene in Unity, which will put them in the same environment to work together.

2.2.5 – Hard Mode Decomposition

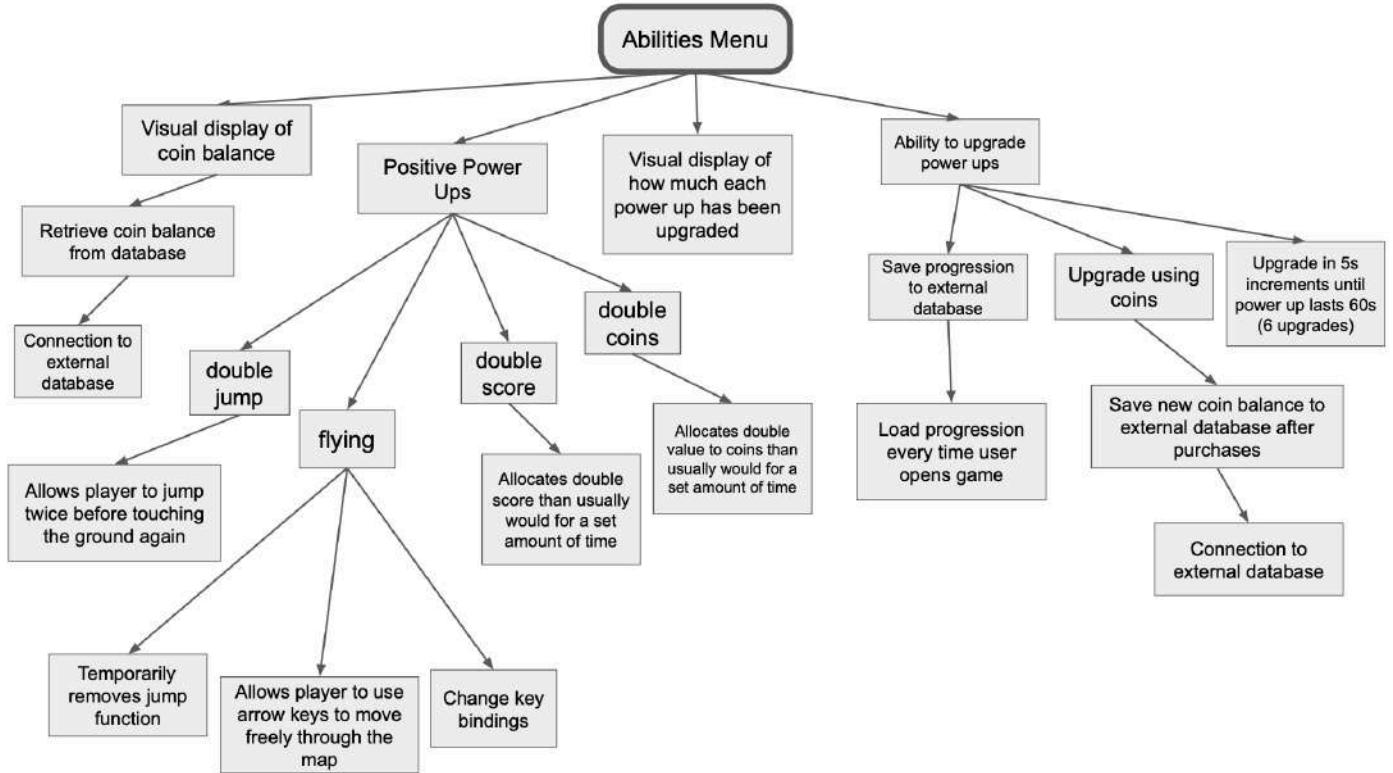


I have separated hard mode into 4 main sections, with a multitude of sub-sections respectively. I have done this in order to make development clearer and simpler, and allow me to be able to work and test different sections of the program as I go, and also to be able to encourage modularisation and reusable components within my project.

I am going to tackle the endless movement first, then obstacles, then the positive power ups, then the negative power ups, then the fast escalation in difficulty, then player death, and finally the pause game aspect. I will do the endless movement aspect first because it is the environment that the other 3 aspects exist in during gameplay. I am doing obstacles next, as this is the driving force of the game and what makes it enjoyable. This is also a reusable component from the easy mode, which can be slightly modified to fit this game mode. I am doing the positive and negative power ups next, as I believe that it will be a good benchmark to test the game, and that aspect will need to be altered when I do the slow escalation in difficulty, which will bring the game together. Lastly will be the pause game function, as it links into the development of the pause menu. I will also bring all of these separate aspects together through implementing them within the same scene in Unity, which will put them in the same environment to work together.

I will be using all of the basic programming structures (iteration, selection and sequence) within both game modes to aid the development of the game, and to make the code as efficient as possible.

2.2.6 - Abilities Menu Decomposition

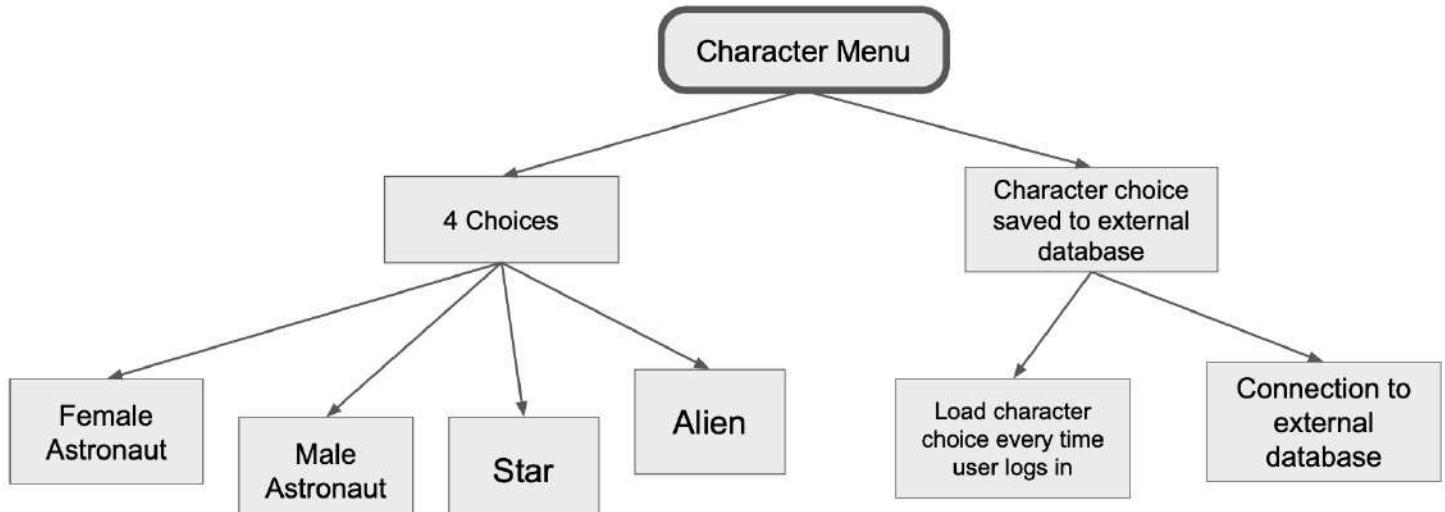


I have split the abilities menu into 4 sections, with 2, 10, 0 and 6 sub-sections respectively. I have done this to break down the development of the abilities menu, and the abilities themselves, into manageable sections. This means that I can start with the visual display of coin balance, power ups, and develop all of them, and then move onto the ability to upgrade power ups and then finally the visual display of how much power ups have been upgraded, without being overwhelmed by doing all of them at once.

I have chosen to do it in this order because I believe that it is the most logical order. I have chosen to do the visual display of coin balance first, as I believe this is essential to be able to test the other aspects of the abilities menu, as it all runs on the in-game currency. This is because the rest of the abilities menu doesn't exist without the power ups themselves also existing first. I have chosen to do the visual display next, as I believe that this has to come before the ability to upgrade, as this is essential in the testing of the ability to upgrade.

Within this, I have decided to use all 3 programming constructs (iteration, selection and sequence) to make my code as efficient as possible. This is because I am to include the most appropriate programming structure in every step of the coding process to make the code as efficient as it can be. For example, I plan to use sequence for the connection to the external database to make sure that it goes as smoothly as possible.

2.2.7 - Character Menu Decomposition

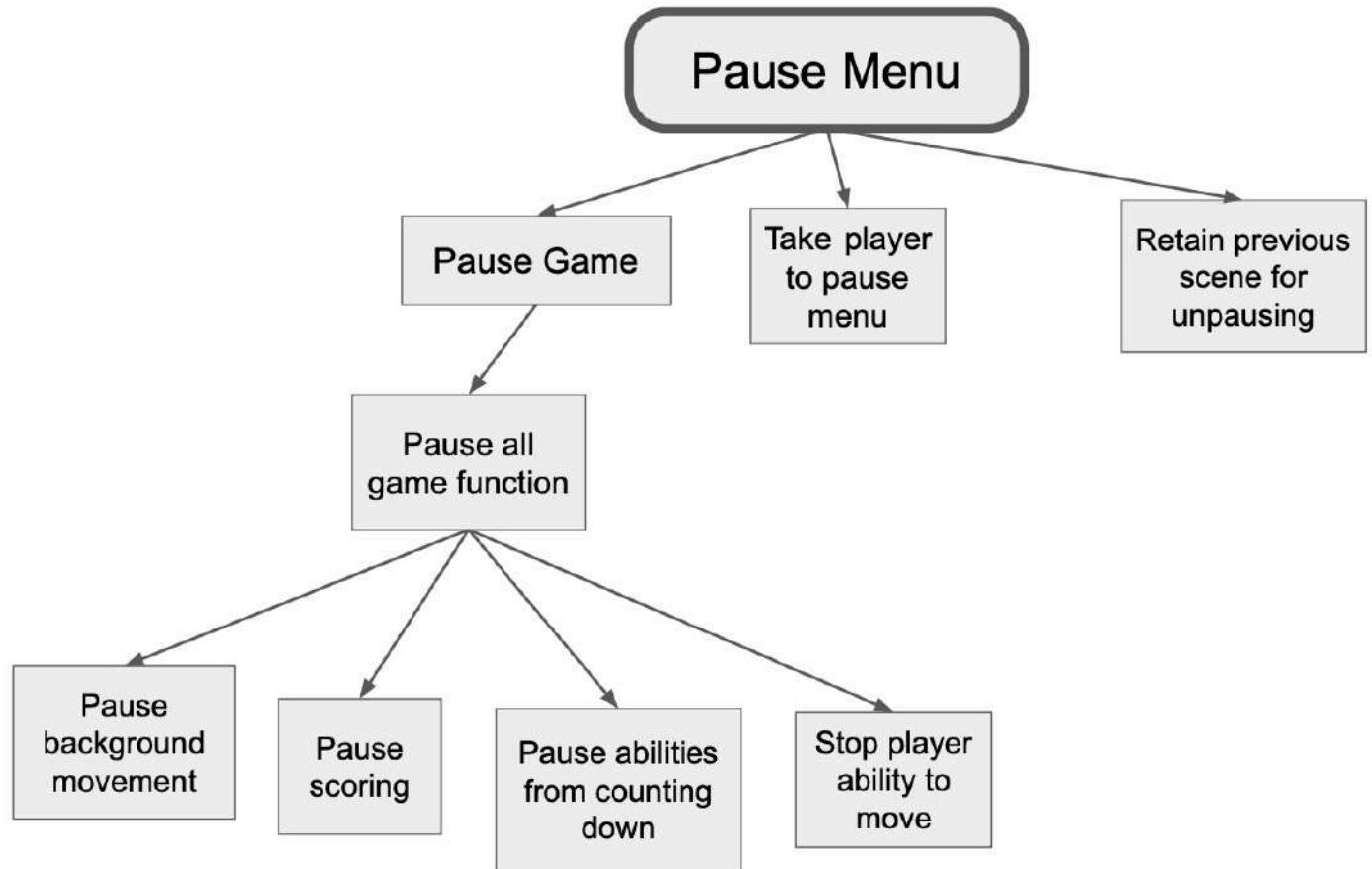


I have split the character menu into 2 sections, with 4 and 2 sections respectively. I have split it like this to separate the part that is purely handled in-game (4 choices) from the part that deals with the external database. I believe that this makes development simpler and easier, and these 2 separate sections can be brought together at the end to form the final solution to my character menu.

I am going to tackle the 4 choices first, as the other part cannot exist without the 4 choices existing first. I am going to tackle each choice in the order shown, from left to right. This makes development easier as the selection of the characters is very similar and aspects from the 1st development of a character selection can be reused for the other 3. After I have assigned variables to the choices and have fully developed them, saving the character choice to the external database will be much easier.

I will be using the programming constructs of selection for the selecting of the 4 choices, as a way to trigger a routine if the select button is pressed on each. I have also chosen to use sequence for the character choice to save to the external database, because it needs to be done sequentially to ensure that it is all saved correctly.

2.2.8 - Pause Menu Decomposition

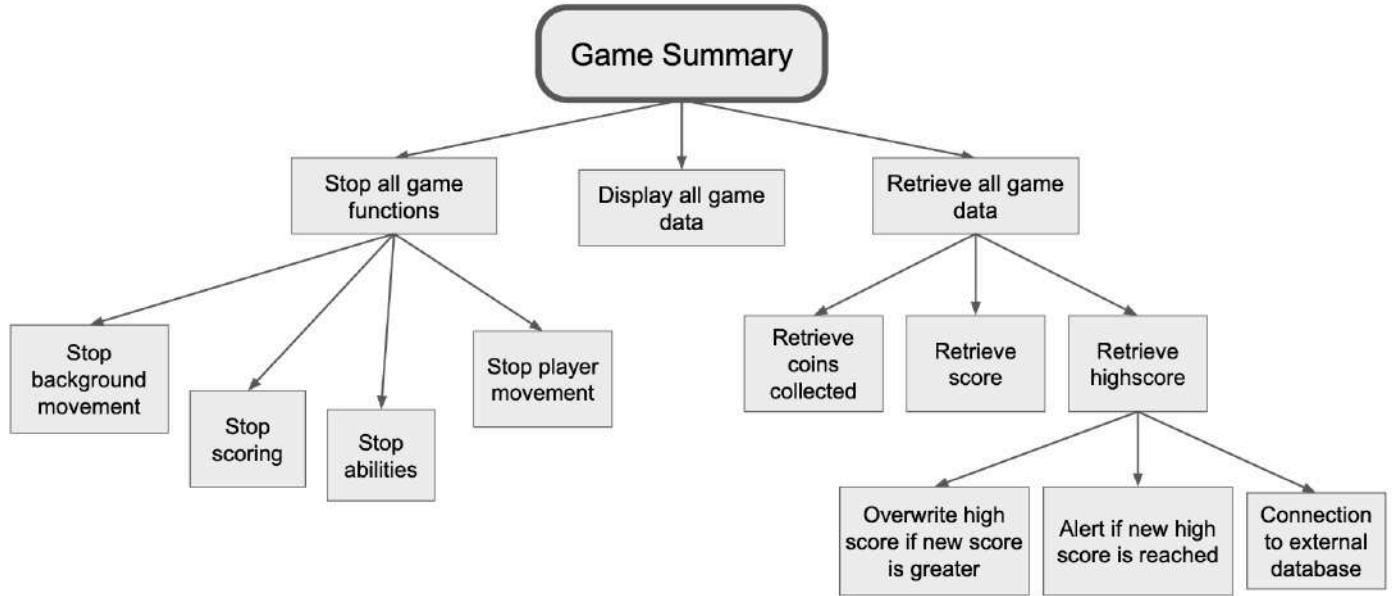


I have chosen to split this into 3 sections, with 6, 0 and 0 sub-sections respectively. I have split it like this because the 'Pause Game' aspect deals with the data to do with the game, whereas the 'take player to pause menu' is the transition between the gameplay and the pause menu, and the retaining previous scene aspect is to do with the visuals of the gameplay.

I am going to tackle the 'Take player to pause menu' first, as this is the environment for the other aspects to exist in. I am then going to implement the 'Pause Game' which pauses all of the game function, as this logically comes before the last step, 'retain previous screen for unpausing', which I will tackle last. This makes development of the solution easier because the aspects, when done in my order, build the foundations for each other, and means that I don't have to go back and fill the gaps when developing.

I have chosen to use the programming construct of sequence, as the most important thing in this menu is that all steps happen in order. This is to ensure that the player's experience is not interrupted, and there is a seamless transition between pausing and resuming the gameplay.

2.2.9 - Game Summary Decomposition



I have chosen to split the game summary screen into 3 sections, with 4, 0 and 6 sub-sections respectively. I have chosen to do this because I believe that it will make development easier to do, as I can develop every single sub-section individually and bring them together to form the final solution.

I have chosen to stop all game functions first, then retrieve all game data, and finally display all game data. I have chosen to do it in this order because the 'stop all game functions' is the foundation of the game ending, and supplements the existence of the other 2 aspects. I have chosen to retrieve game data next, as this is the next logical step; I cannot display game data before I have retrieved it, and I can't retrieve the game data if the game hasn't stopped. Therefore, the last step to develop is the display of all game data.

I have decided to use the programming construct of sequence, as I believe that the most important aspect of the game summary is that all steps are completed sequentially, because they all supplement each other.

2.3 - Main Menu Design

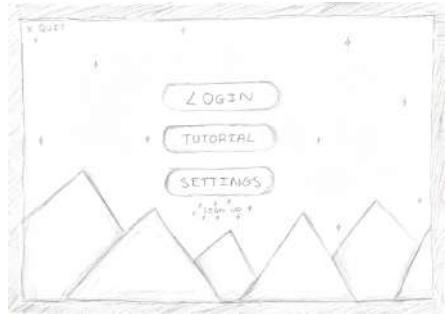
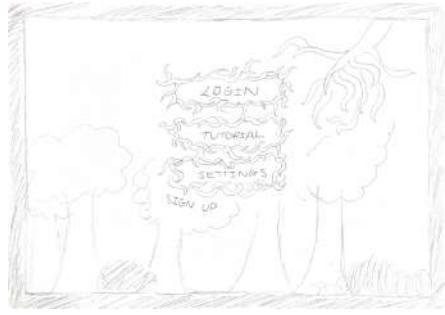
2.3.1 - Initial Designs

These are the 3 initial designs for my main menu. This is what users will see when they first open the game, and will have 2 options:

- Login
- Sign up

I have created 3 initial designs for this main menu, pictured below.

Initial Idea No.	Idea	Design
------------------	------	--------

1	<p>This is a space themed design, with the moon pictured in the bottom third of the picture. There is an alien, an astronaut's helmet & stars in the sky (space) with the 4 options outlined above, with the sign up aspect of the menu being signposted smaller at the top of the login button</p> <p>It also has a grey border surrounding it</p>	
2	<p>This is a mountain range themed menu, with a black border. There are stars in the sky and it would have a sunset gradient behind the mountains, which themselves would be a grey-black gradient, starting from the top of the mountains</p>	
3	<p>This is a forest themed menu, with a brown border. The buttons themselves are surrounded by branches and the sign up button being signposted within a tree design</p> <p>There are trees in the bottom half of the menu, with bushes depicted in the bottom quarter of the design.</p>	

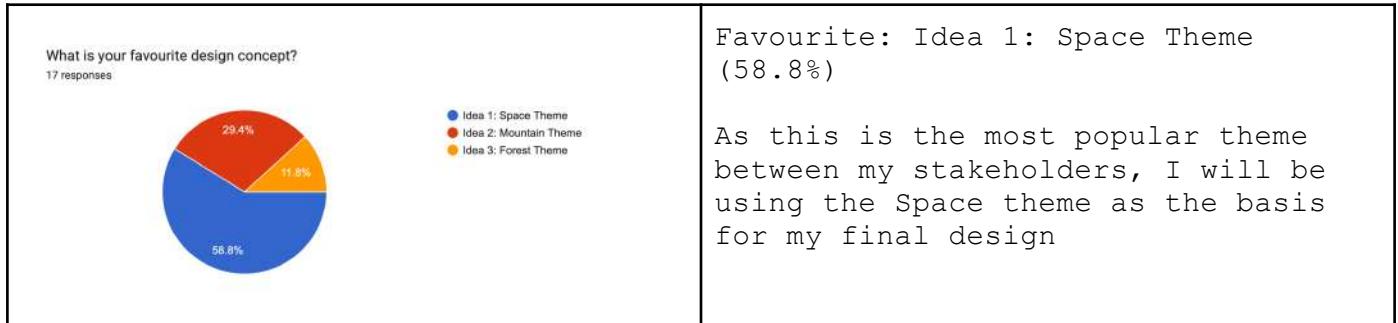
2.3.2 - Stakeholder Review

I sent out a form to a group of people that represent the stakeholders for my project, asking them their opinions on the designs I have created

- Firstly, I asked if they agreed or disagreed with a statement
 - 'How much do you agree with this statement: '[Insert Idea] is the most creative & understandable menu idea'
- Then I asked what their favourite idea was out of the ones presented to them

Questionnaire Results

Question & Result	Comments												
<p>How much do you agree with this statement: 'Idea 1: Space Theme is the most creative & understandable menu idea'</p> <p>17 responses</p> <table border="1"> <thead> <tr> <th>Agreement Level</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0%</td> </tr> <tr> <td>2</td> <td>5.9%</td> </tr> <tr> <td>3</td> <td>11.8%</td> </tr> <tr> <td>4</td> <td>47.1%</td> </tr> <tr> <td>5</td> <td>35.3%</td> </tr> </tbody> </table>	Agreement Level	Percentage	1	0%	2	5.9%	3	11.8%	4	47.1%	5	35.3%	<p>Average Score: 4.1/5</p> <p>I believe that this first idea received the best score because of it's better clarity in design than the other 2, but is not a full 5/5 because it is not the most creative, and can be seen as plain due to the overwhelmingly dark nature of the background</p> <p>I plan to mainly incorporate features from this idea to my final design</p>
Agreement Level	Percentage												
1	0%												
2	5.9%												
3	11.8%												
4	47.1%												
5	35.3%												
<p>How much do you agree with this statement: 'Idea 2: Mountain Theme is the most creative & understandable menu idea'</p> <p>17 responses</p> <table border="1"> <thead> <tr> <th>Agreement Level</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>5.9%</td> </tr> <tr> <td>2</td> <td>17.6%</td> </tr> <tr> <td>3</td> <td>11.8%</td> </tr> <tr> <td>4</td> <td>47.1%</td> </tr> <tr> <td>5</td> <td>17.6%</td> </tr> </tbody> </table>	Agreement Level	Percentage	1	5.9%	2	17.6%	3	11.8%	4	47.1%	5	17.6%	<p>Average Score: 3.5/5</p> <p>I believe that this is the next highest scoring menu idea because it's very creative & aesthetically pleasing as well as clear, however it is not as clear as Idea 1, making it the 2nd best menu idea.</p> <p>I plan to slightly incorporate features from this idea to my final design</p>
Agreement Level	Percentage												
1	5.9%												
2	17.6%												
3	11.8%												
4	47.1%												
5	17.6%												
<p>How much do you agree with this statement: 'Idea 3: Forest Theme is the most creative & understandable menu idea'</p> <p>17 responses</p> <table border="1"> <thead> <tr> <th>Agreement Level</th> <th>Percentage</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>11.8%</td> </tr> <tr> <td>2</td> <td>29.4%</td> </tr> <tr> <td>3</td> <td>41.2%</td> </tr> <tr> <td>4</td> <td>11.8%</td> </tr> <tr> <td>5</td> <td>5.9%</td> </tr> </tbody> </table>	Agreement Level	Percentage	1	11.8%	2	29.4%	3	41.2%	4	11.8%	5	5.9%	<p>Average Score: 2.7/5</p> <p>I believe that this is the lowest scoring menu idea because, although it is very creative, with the potential for a large variation of colours through the foliage in the design, the design itself is not very cohesive & the extensive branches make the writing of the menu options unclear. However, one thing I believe does make it clear is the clear signposting of the 'sign up' button</p> <p>I plan to incorporate very few aspects from this idea to my final design</p>
Agreement Level	Percentage												
1	11.8%												
2	29.4%												
3	41.2%												
4	11.8%												
5	5.9%												



2.3.3 - Evaluation of Review

From this stakeholder review, I have concluded that Idea 1: Space Theme is the most popular design idea for the main menu. However, the other 2 designs do have their own merits & aspects that the stakeholders found to either be creative or clear with the other 2 ideas.

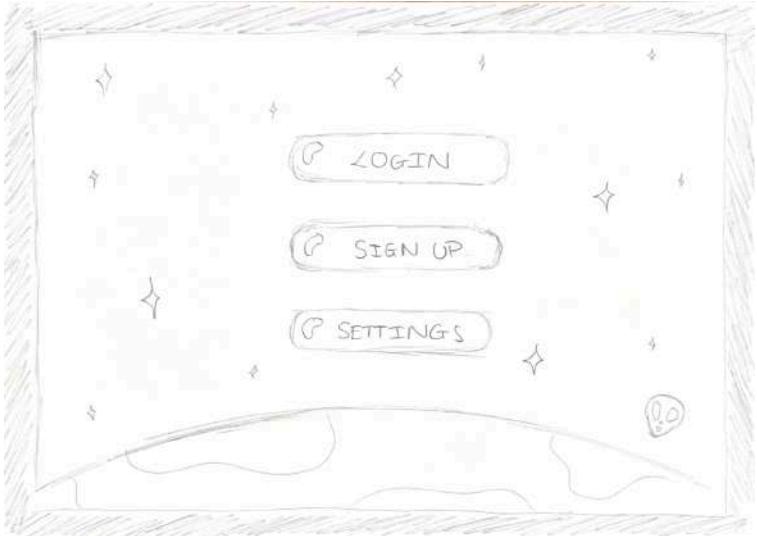
From Idea 1 (Space), I will keep the main idea of space, however I will add & change aspects of it, based upon the other designs

- From Idea 2 (Mountains): I will make the sky of the space have just stars in it, of varied sizes, just like that of the mountain sky
- From Idea 3 (Forest): I will clearly signpost the 'Sign Up' button on screen, so that it is clearly visible & user friendly. I will also make the Space Theme more colourful, by changing the moon on the bottom 3rd of the menu to a more colourful planet, Earth.

To improve clarity of idea 1, I have also decided to move the tutorial option button to be a post-login option, for a more coherent program. This is because you cannot play the game without first logging in or signing up, so the tutorial button being before the user logs in is very redundant & slightly confusing, which is not beneficial, as my success criteria outlines the need for clear signposting of features within the game.

2.3.4 - Final Design

Based upon my previous reflections, I have created a final design for my main menu, which all other menu designs will be based off of.



And I have then created a digital version of this design, which will be used in-game:



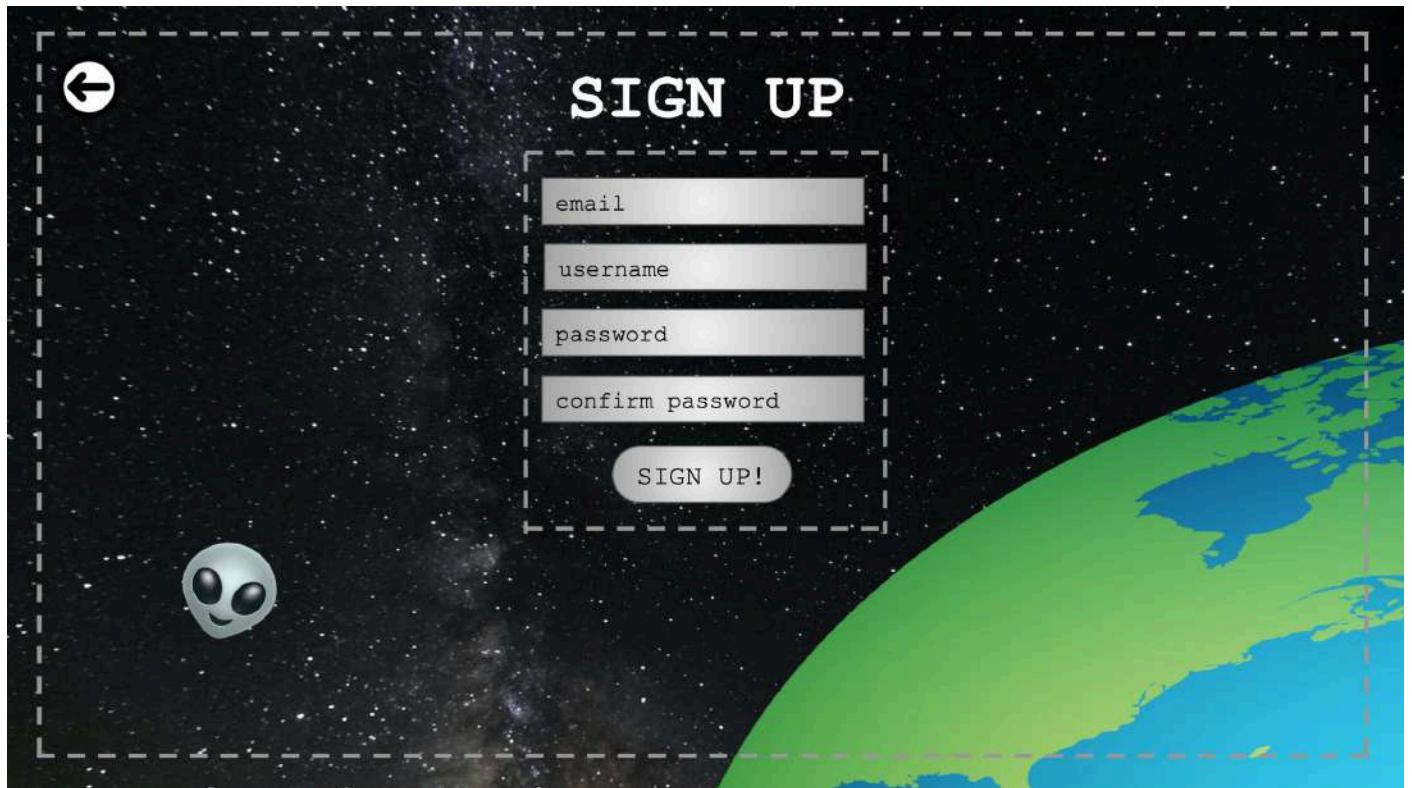
Image Sources:

- Space.com, Clipartpng.com, Symb1.cc

2.4 - Menu Designs & Icons

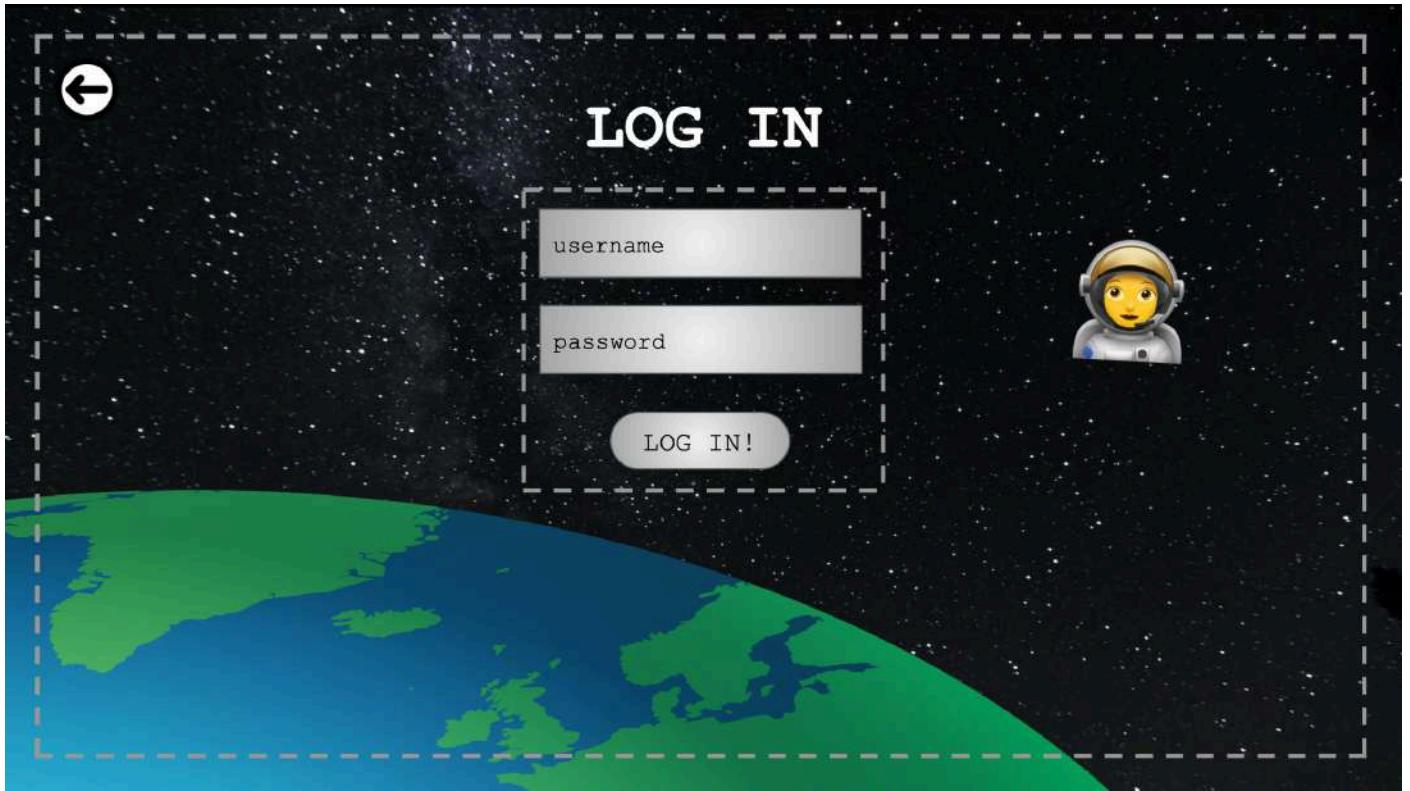
2.4.1 - Sign Up Menu

I have created a sign-up menu based upon my main-menu design. By doing this, I am making sure that my whole game's design is coherent, but not exactly the same.



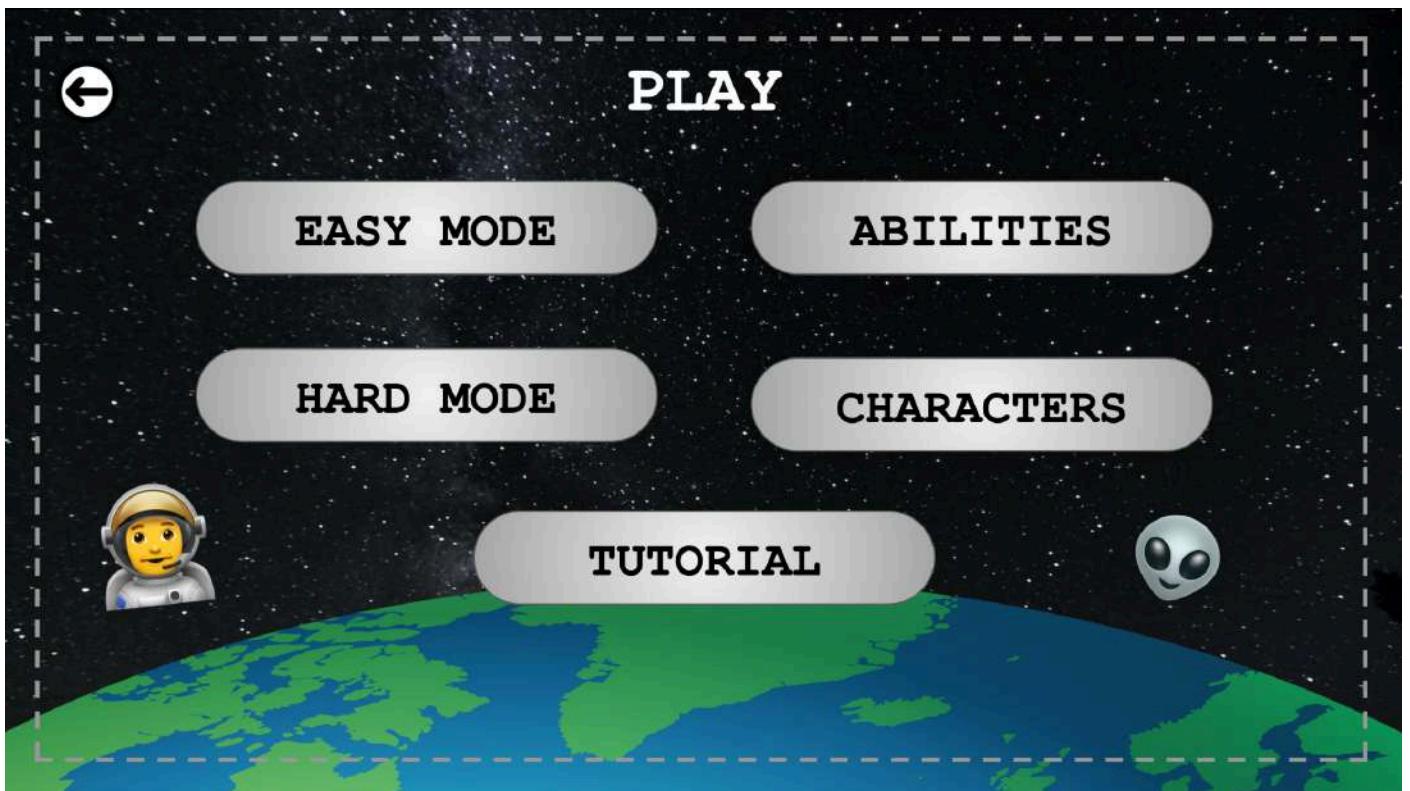
2.4.2 - Login Menu

I have created a login menu based upon my main-menu design. By doing this, I am making sure that my whole game's design is coherent, but not exactly the same.



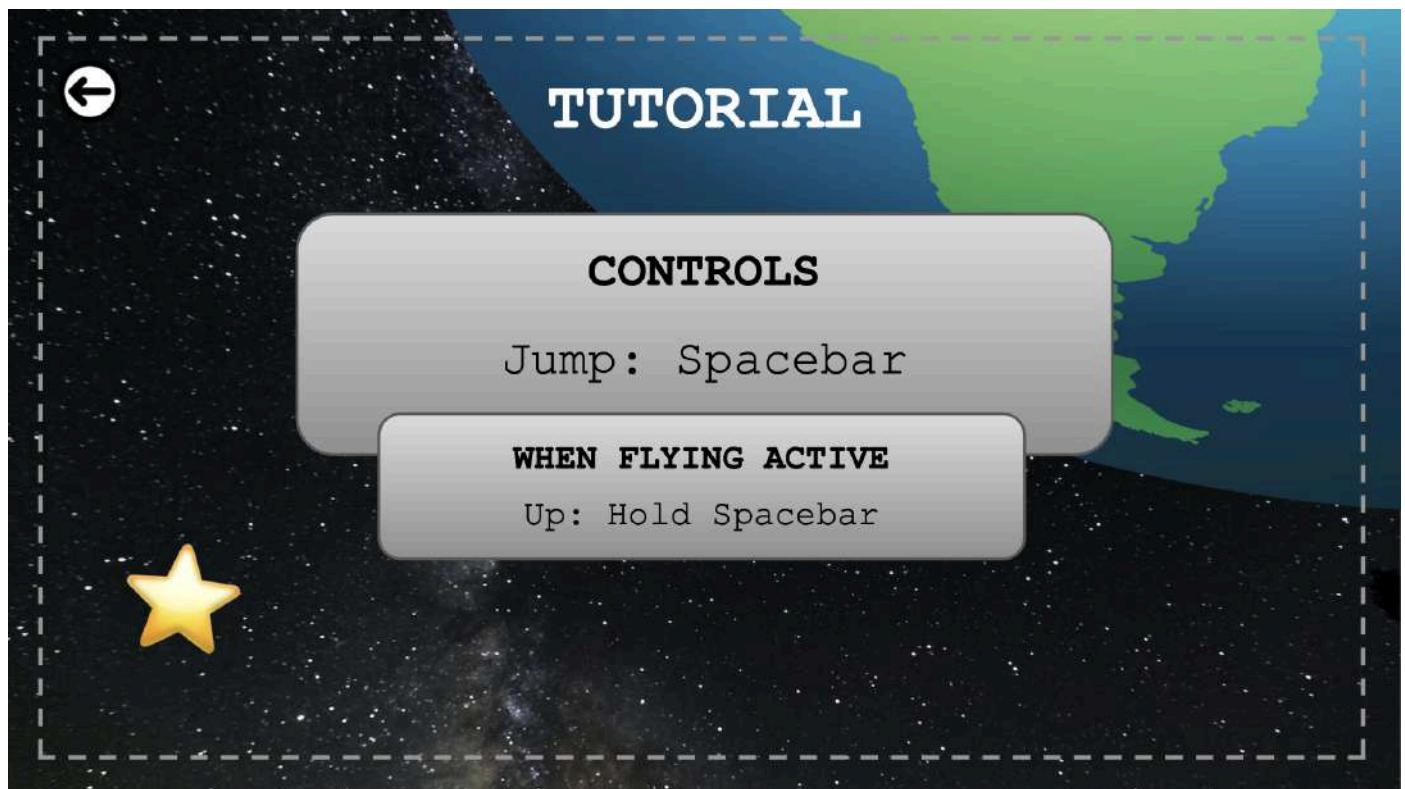
2.4.3 - Play Menu

I have created a play menu based upon my main-menu design. By doing this, I am making sure that my whole game's design is coherent, but not exactly the same.



2.4.4 - Tutorial Screen

I have created a design for the tutorial screen. The controls of AstroSprint won't be very complicated, so the screen itself is very basic. Design is made to be coherent, but not identical to others.



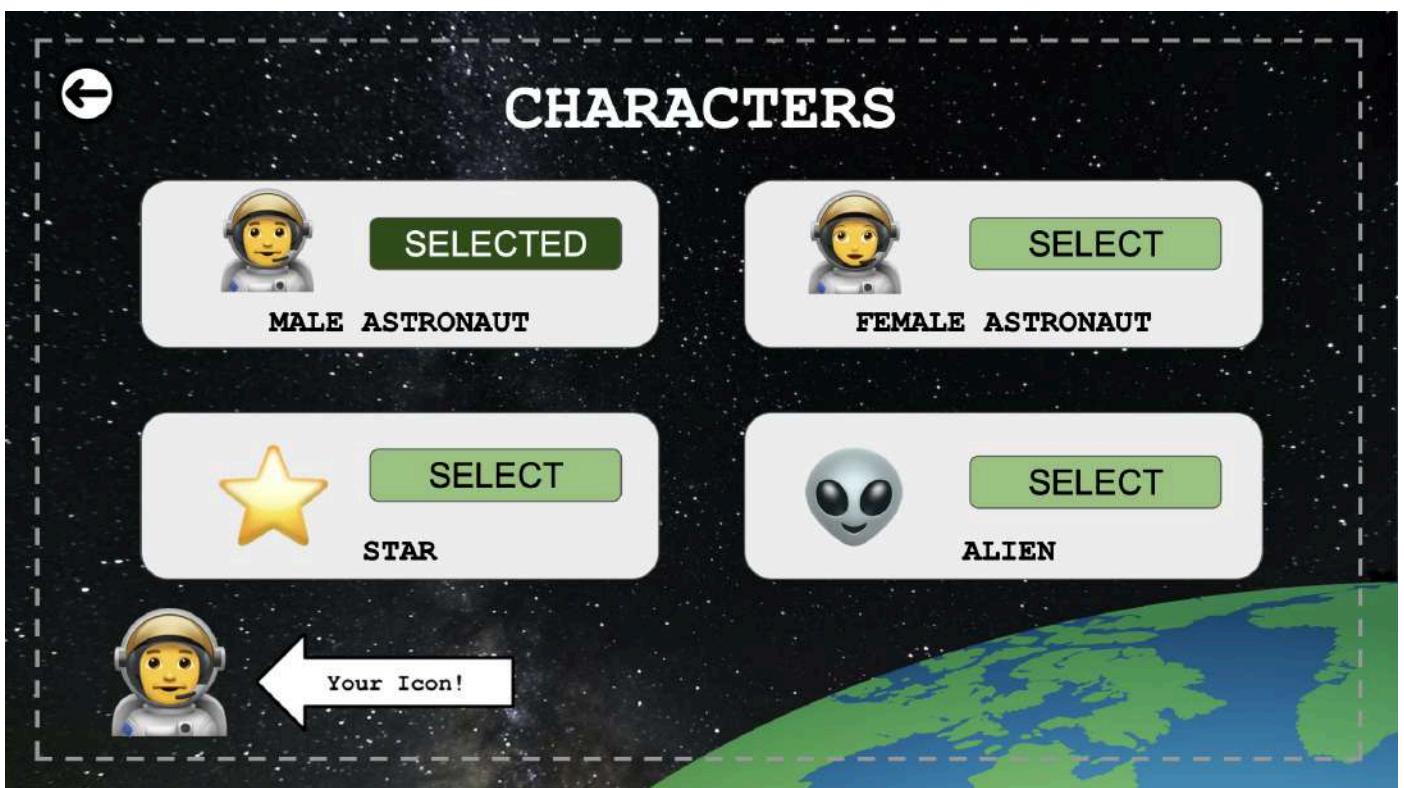
2.4.5 - Abilities Menu

This is my design for the abilities menu. The design is made to be coherent, but not identical to others. The actual coin balance will be a variable shown in-game.



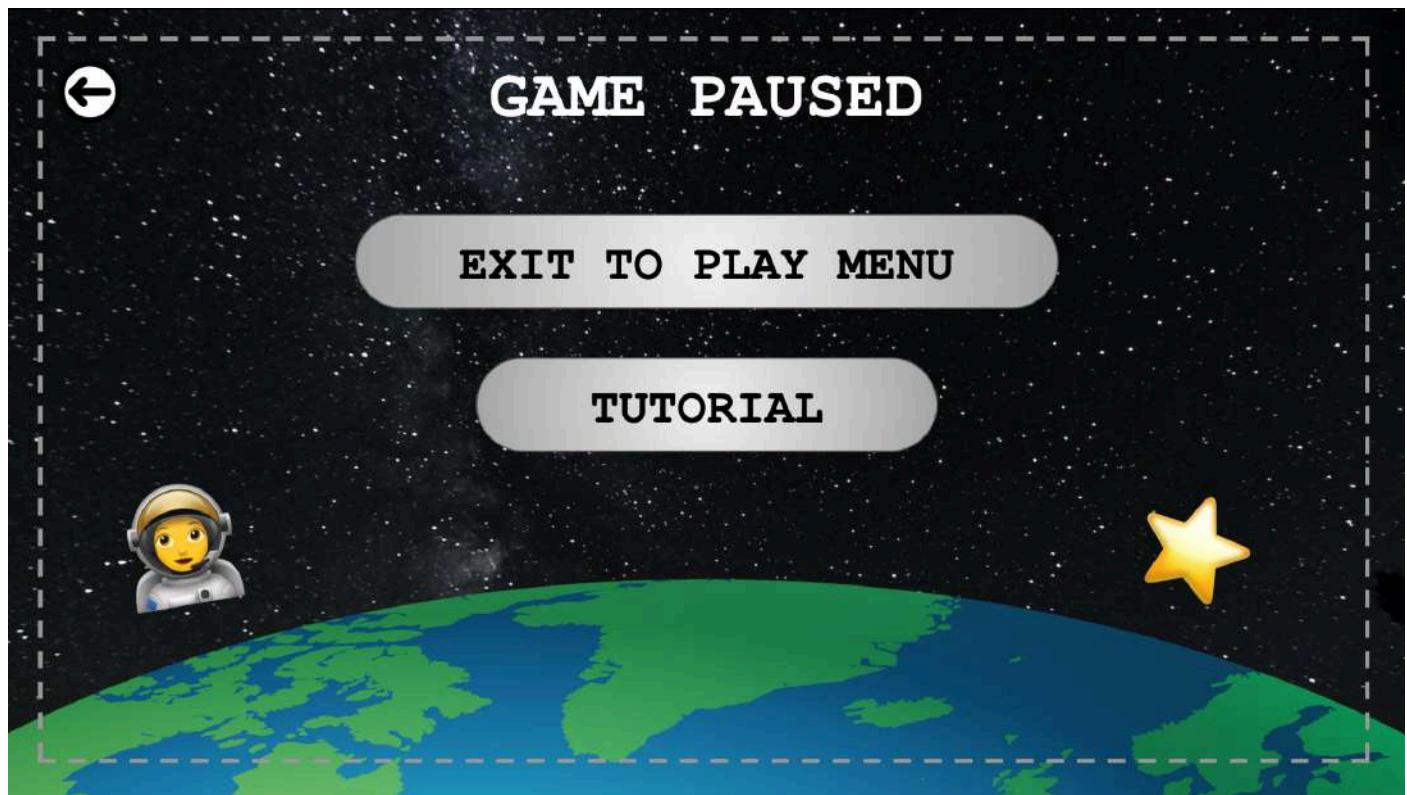
2.4.6 - Characters Menu

This is my design for the characters menu. The design is made to be coherent, but not identical to others.



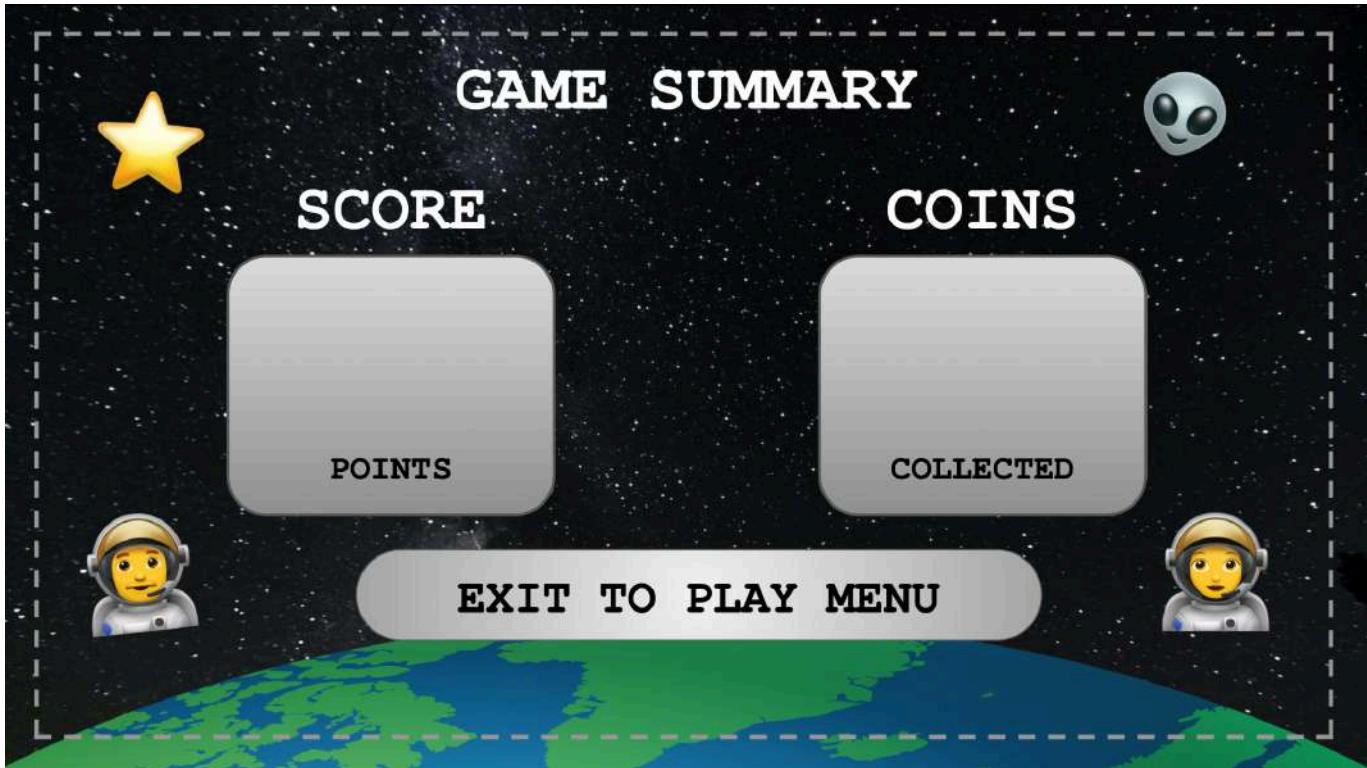
2.4.7 - Pause Menu

This is my design for the pause menu. The design is made to be coherent, but not identical to others.



2.4.8 - Game Summary Screen

This is my design for the game summary screen. The design is made to be coherent, but not identical to others. The information of the score & coins collected will be shown after actual gameplay.



2.4.9 - Icons

Icon	Name	Use	Image Source
	Double Jump	A representative of the double jump power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.rolimons.com/gamepass/9230676
	Double Points	A representative of the double points power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.roblox.com/game-pass/11393260/PRE-RELEASE-Double-Points
	Fly	A representative of the flying power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.roblox.com/game-pass/2596638/Fly

	Double Coins	A representative of the double coins power up, to be used in the Easy & Hard Mode gameplay and the abilities menu	https://www.roblox.com/ko/game-pass/103796180/Gamepass-Icon
	Male Astronaut Character	A representative of the male astronaut character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/2010/smileys-people/man-astronaut
	Female Astronaut	A representative of the female astronaut character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/2106/smileys-people/woman-astronaut
	Star Character	A representative of the star character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/586/animals-nature/white-medium-star
	Alien Character	A representative of the alien character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/34/smileys-people/alien
	Darkness	A representative of the darkness power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/close-eyes_4264667
	Blurry Screen	A representative of the blurry screen power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/dizzy_2302669

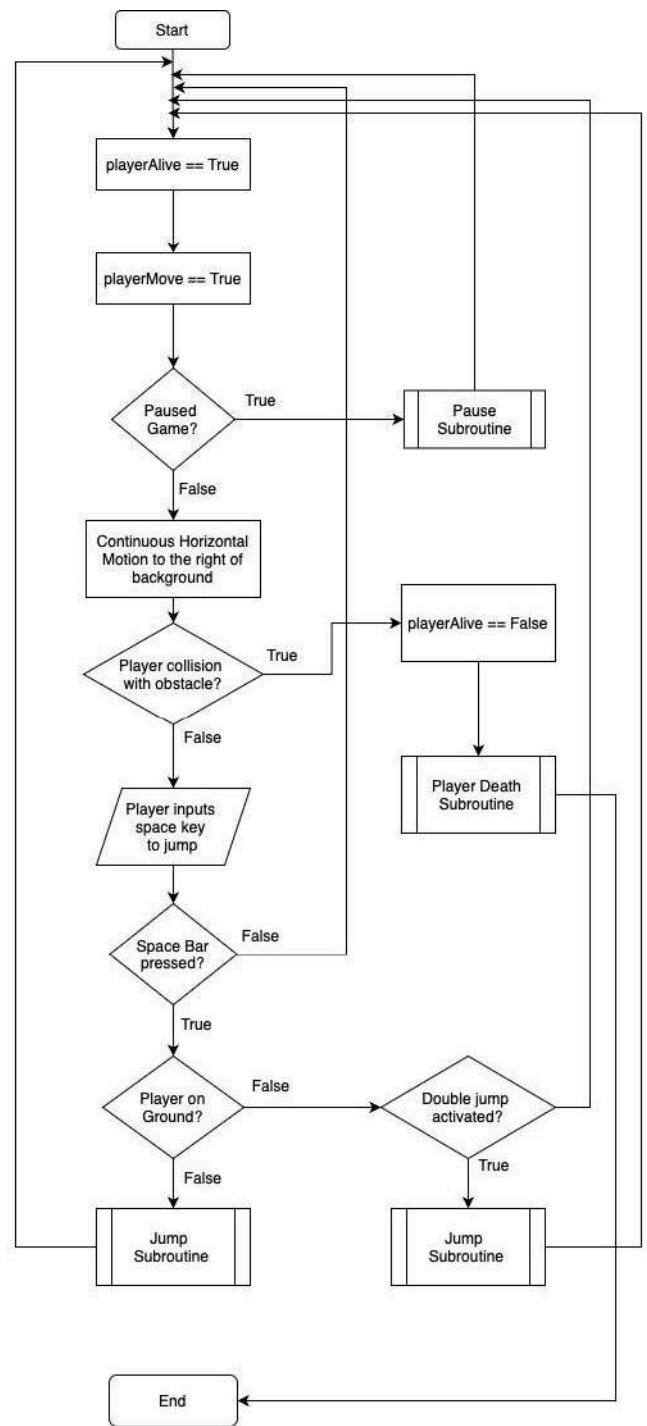
	Half Score	A representative of the half score power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/fraction_9845747
	Inverted Colours	A representative of the inverted colours power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/invert_7746578
	New Highscore Icon	Appears on the game summary screen when a new high score is achieved	https://www.flaticon.com/free-icon/high-score_13446444
	Back Button	The button that allows players to go backwards to a previous menu/screen in-game	https://www.flaticon.com/free-icon/back_318292

2.5 - Key Process Flowcharts

2.5.1 - Movement Flowchart

This flowchart describes the movement of the player and the background in a very basic sense.

The idea of there being horizontal movement (as the background has movement, not the player themselves, is explored here, as well as the idea of the player jumping, the player being able to jump twice if double jump is active and the player being able to cause the game at any time.



2.5.2 - Jumping Subroutine Flowchart

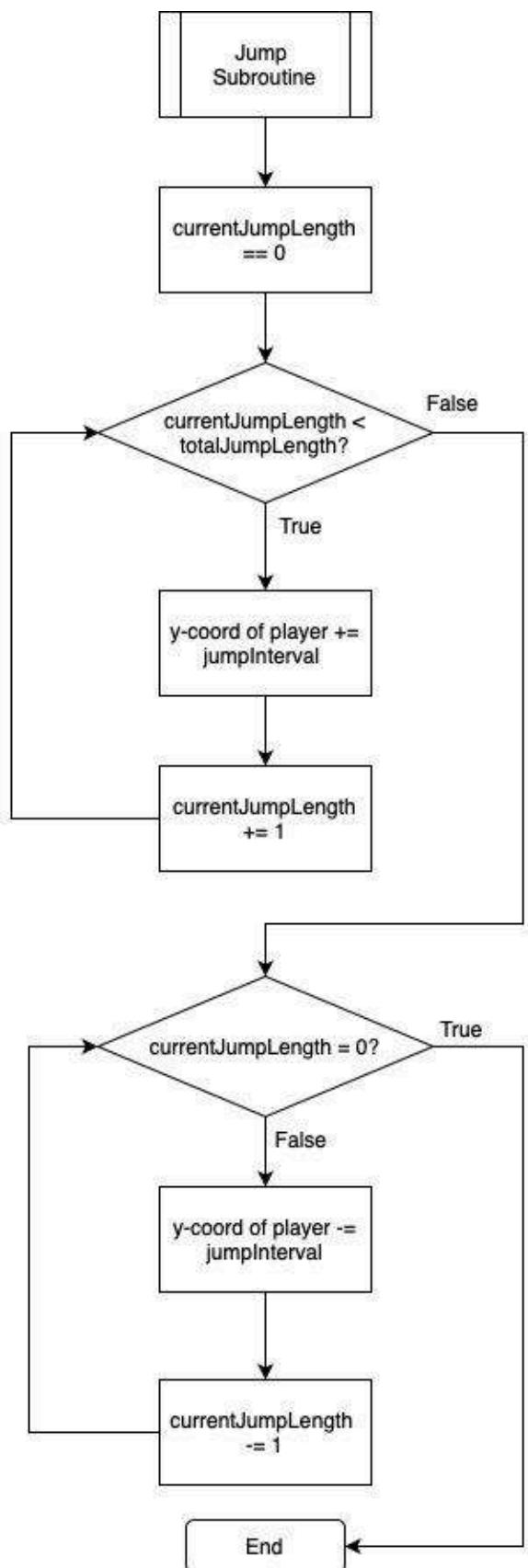
The subroutine for jumping is shown to the right.

This subroutine explains that the y-coordinate of the character is incrementally increased at every refresh of the frames in Unity.

I have done it this way to ensure that the movement of the player when jumping isn't static, as if they are instantly transformed from one coordinate to another. The increment introduced by 'jumpInterval' prevents this, so in-game the player's y-coordinate will be slowly increasing to the final jump height.

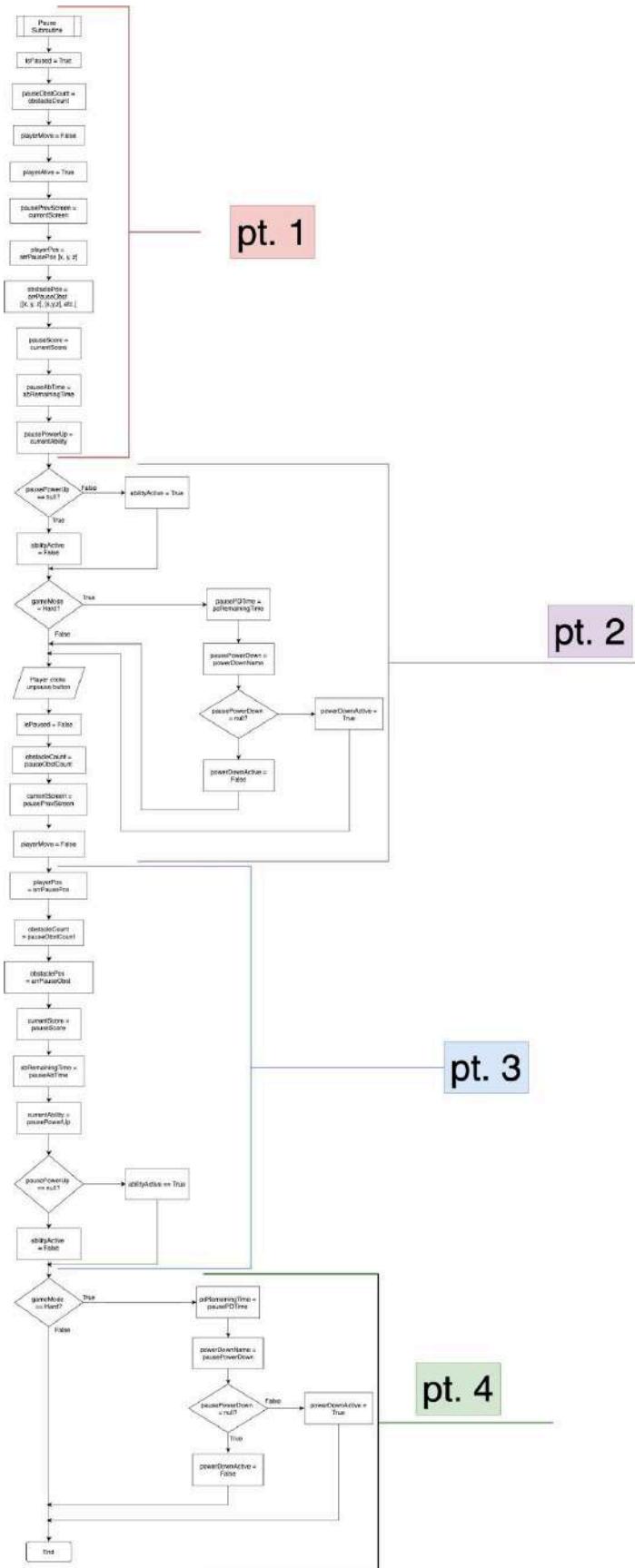
The latter part of the subroutine is there to undo the jump that is created by the first half of the subroutine. To ensure consistency, the player jumps down with the same interval until they are back at their original y-coordinate (the ground)

There is a use of a condition controlled loop in both the jumping up and fall back down of the player, where the subroutine only ends if the currentJumpLength (a variable that describes the player's displacement from the ground) is equal to the jump height when going up, and equal to zero when coming back down.



2.5.3 - Game Pause Flowchart

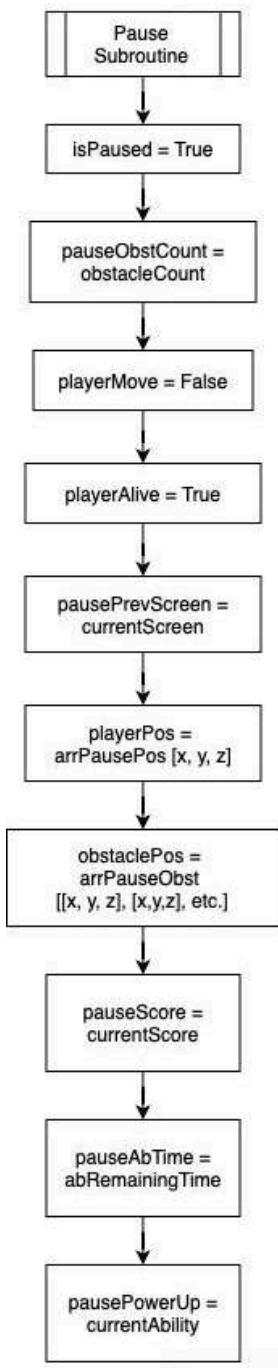
The entire flowchart for pausing the game is shown below. Below this diagram, the flowchart has been split into smaller parts (as denoted on main diagram) for justifications & ease of reading.



Part 1 of Pause Subroutine

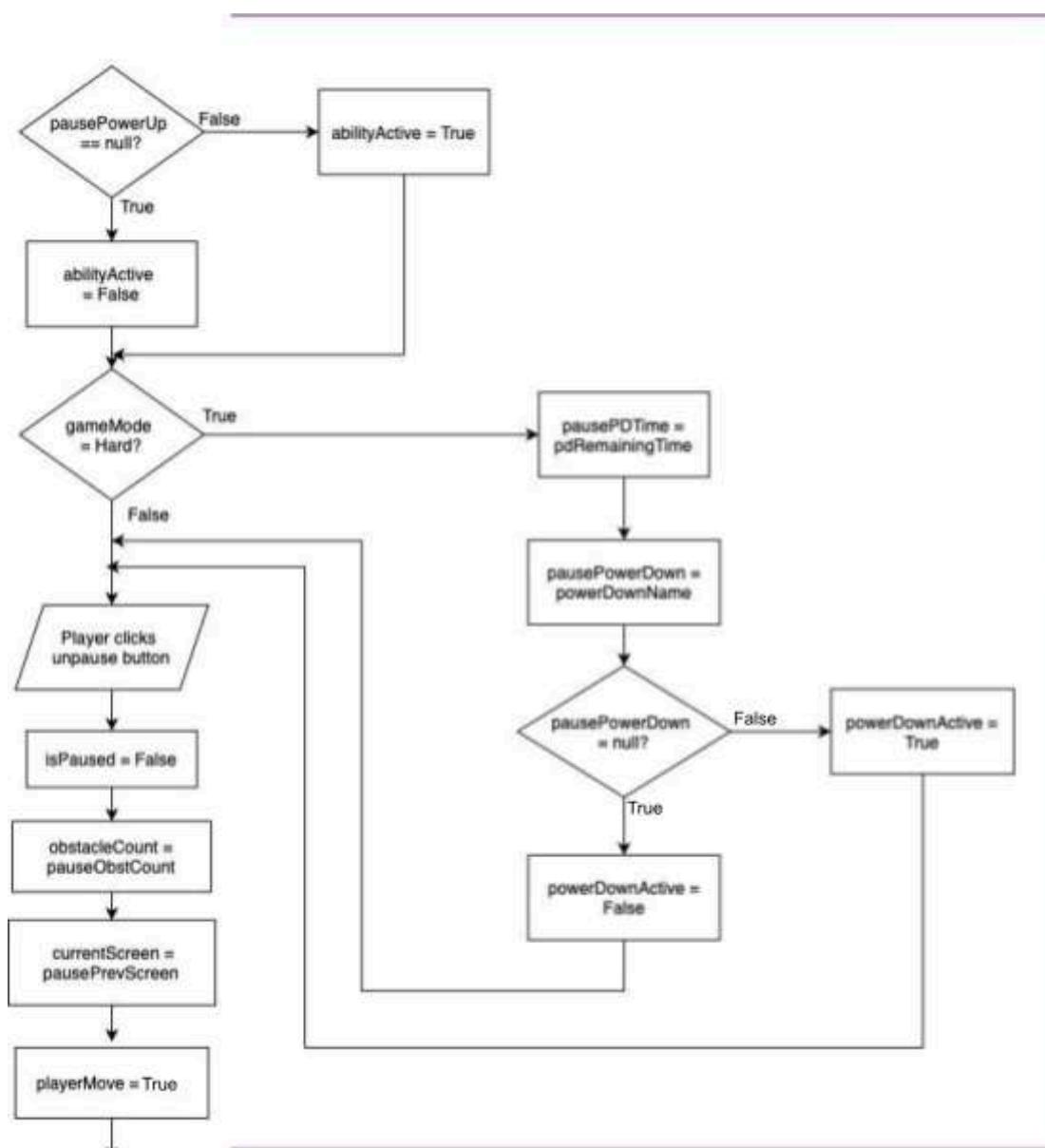
This is the first part of the subroutine used to pause the game.

This first aspect of the subroutine is just getting the current stats of the game and putting them into new variables that will not be affected by the scoring system of the game, and to prevent the running down of abilities or power downs when the game is paused.



pt. 1

Part 2 of Pause Subroutine



pt. 2

This is the second part of the subroutine used to pause the game.

This section firstly clarifies whether a power up is active or not, depending on whether a value is passed into the variable 'pausePowerUp' (outlined in part 1 of the subroutine). I have done it this way to ensure that the clock running down on the ability in the time it takes to execute the subroutine doesn't affect the un-pausing of the game and allows the player to continue to have their ability.

The next section of pt.2 of this subroutine is checking if the game mode is hard, and if it is then the time left & the name of any power downs are recorded in variables to ensure that when the player unpauses they return to the game in the same state as it was when the game was originally paused.

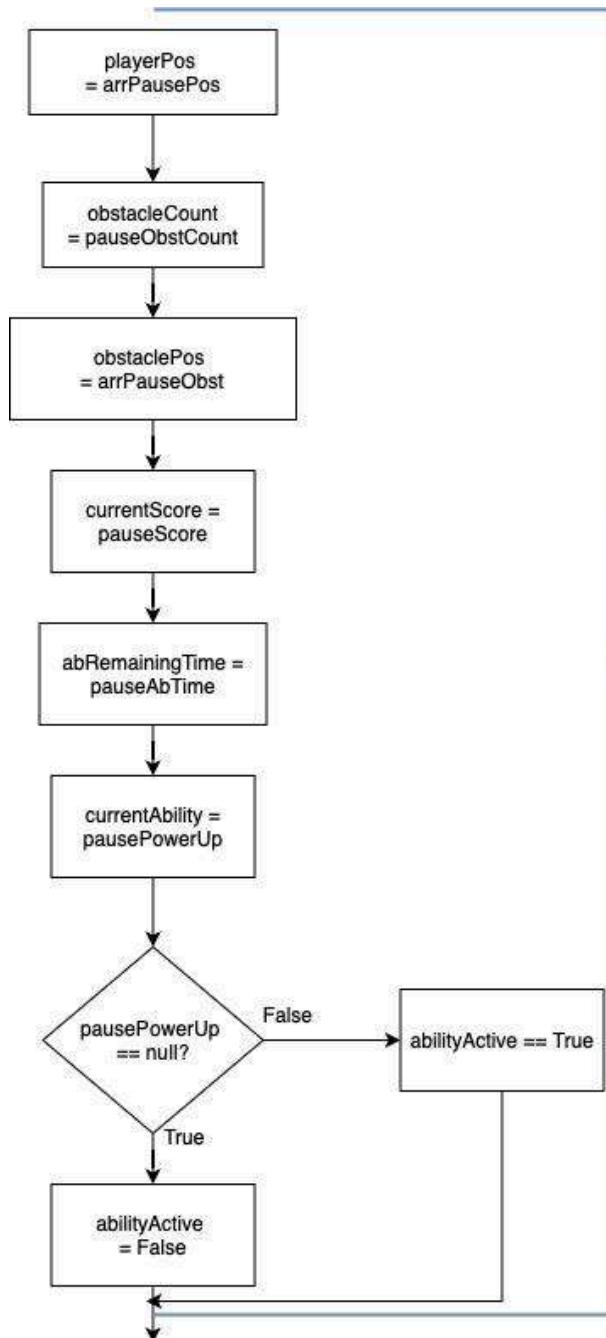
The next section of this part of the pause subroutine is the un-pausing of the game. This section takes the 'pause' variables that were originally assigned values from the game the moment it was paused, and reassigns them to the variables that are used in-game. I have done this to ensure that the game is kept in the same state for the user as when it was originally paused. This section also changes the current screen to the game screen, changing it from the pause screen.

Reassigning the variables to the ones used in-game means that when the game is paused:

- No ability times are run down
- The player cannot die
- No power down times are run down
- No obstacles disappear
- No obstacles can collide with the player

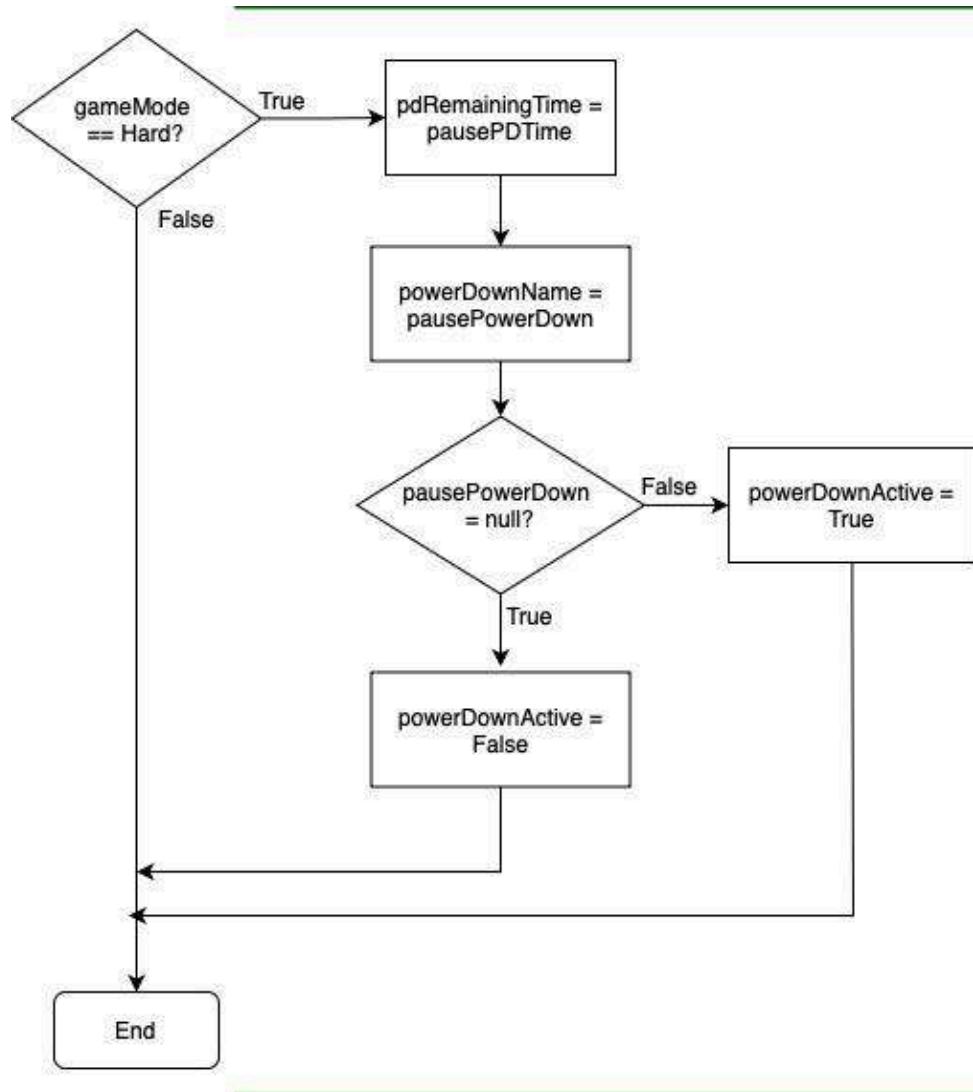
Part 3 of Pause Subroutine

The 3rd part of this subroutine continues to return the game to the state it was in before it was paused. It continues to ensure a consistent smooth running of the game.



pt. 3

Part 4 of Pause Subroutine



pt. 4

The final aspect of this subroutine is the un-pausing of any power downs if the game is in hard mode. If the game mode is easy, then these steps are bypassed and the subroutine ends. If the game mode is hard then the in-game variables are reassigned to be the 'pause' variables that held the value of the in-game variables when it was paused. I have done this to make sure that no power downs are run down and the game is unchanged when it is unpause. This ensures that the game runs smoothly and consistently.

2.5.4 - Coin Generation Flowchart

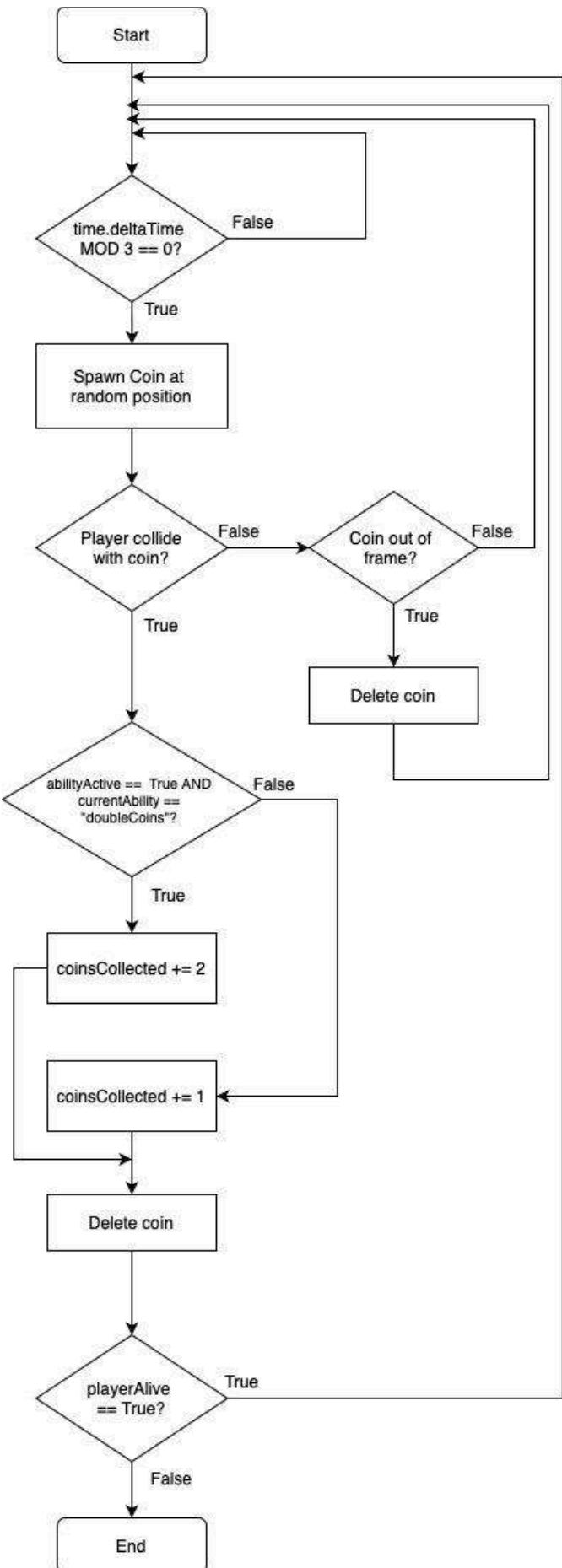
This is the flowchart for the generation of coins in-game. In AstroSprint, coins are in-game currency that are generated every 3 seconds in a random position on the map. Players can collect coins to spend on upgrading abilities.

In this flowchart, it does a modulus division (which provides the remainder) and if the remainder is 0, then the time is a multiple of 3, this means that a coin will only spawn every 3 seconds. I have made it spawn a coin every 3 seconds because I believe that it will prevent the system from being overloaded from all of the coin generation. At the same time, a big time difference between the coins being generated would make the game less enjoyable. The 3 second gap between spawning is the balance between these 2 factors.

The rest of this subroutine then spawns the coin at a random position, then waits until the player collides with the coin. If the player doesn't collide with the coin and it isn't out of frame there is a condition controlled loop until either the player collides with the coin or the coin leaves the frame. At this point, if the coin is out of frame it gets deleted without increasing the player's coins collected.

If the player does collide with the coin, the player gains 1 coin collected if the double coin ability isn't active, and gains 2 coins collected if it is. The coin is then deleted.

As long as the player is alive, the loop repeats. This ensures that the player statistics are not altered after death.



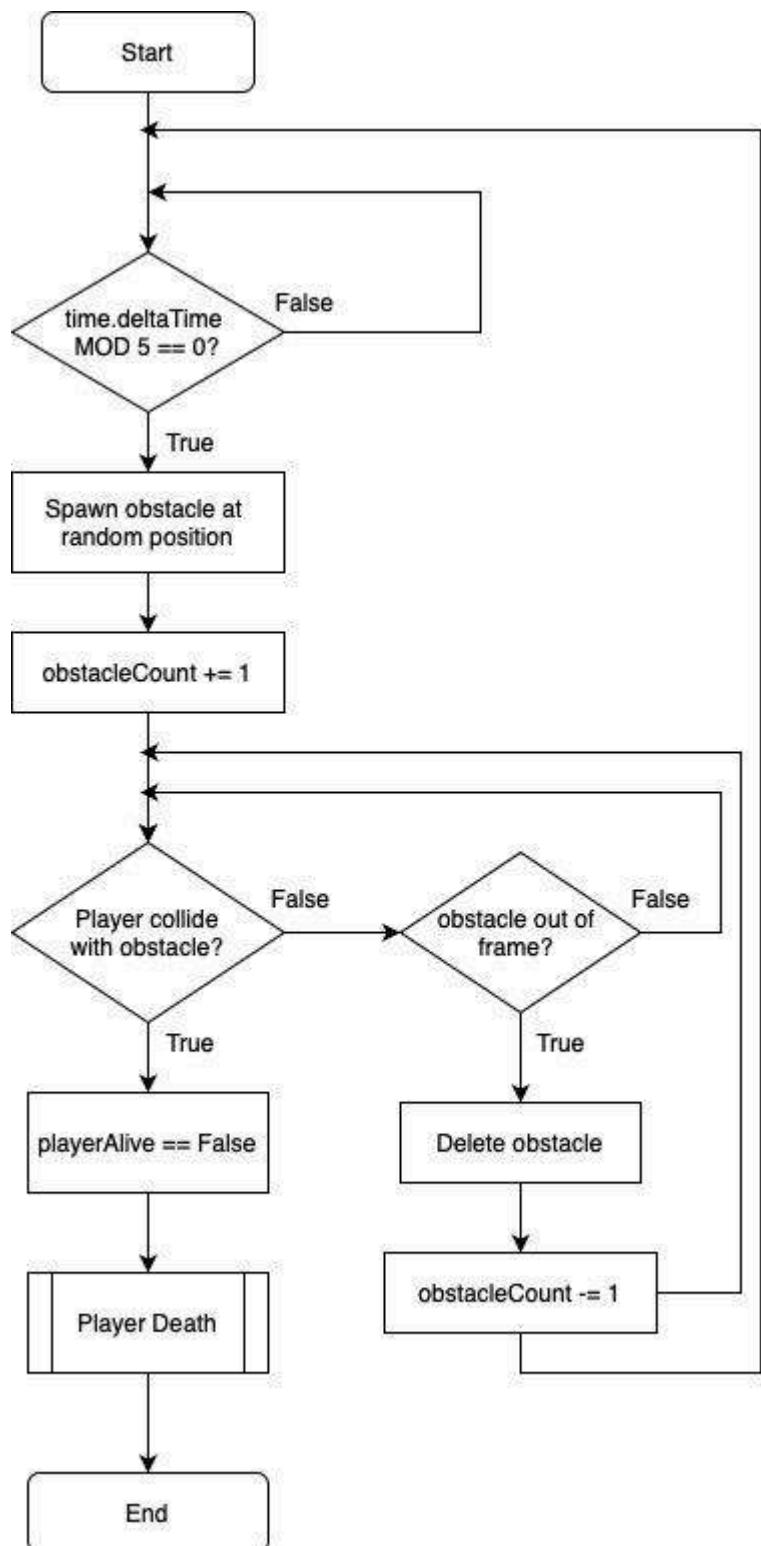
2.5.5 - Obstacle Generation Flowchart

This flowchart describes how obstacles are generated in-game. There is a modular division of the time to ensure that obstacles are generated every 5 seconds, as if the remainder of the time MOD 5 is equal to 0, then the time is a multiple of 5.

I have done this because I believe that it ensures that the game isn't too difficult with an overwhelming amount of obstacles, but at the same time isn't too easy with there being barely any obstacles.

The flowchart then goes on to describe the spawning of an obstacle, and adds 1 to the obstacle count. It then goes into a selection, where if the player collides with the obstacle, they die, but if they don't, there is a condition controlled loop until either the obstacle is out of frame (in which case it is deleted) or until the player hits the obstacle and dies.

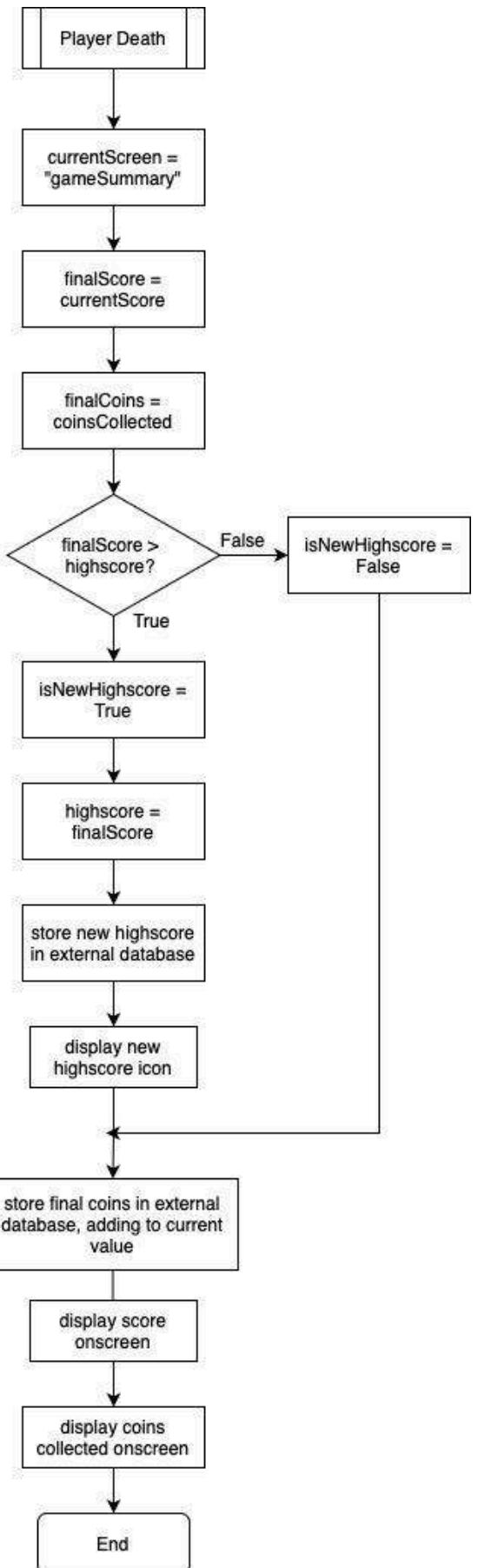
This means that the player only has 1 life, which I have done for simplicity of the game. It was also a feature within the endless runner games that I analysed previously.



2.5.6 - Player Death Subroutine Flowchart

This subroutine describes what happens when a player dies. A player death can only occur after being hit by an obstacle, as it is an endless runner game, which means that the game will continue until the player dies.

After the player dies, the screen changes from the game to the game summary screen. Then the statistics from the game are collected, any new highscores are alerted to the user and then saved in an external database.

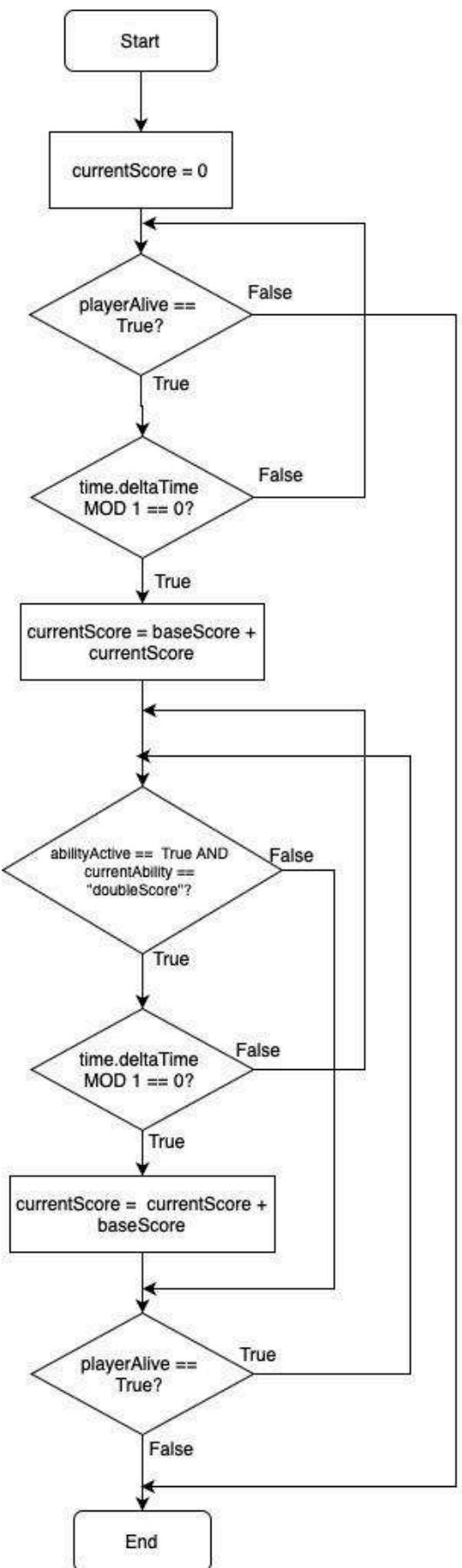


2.5.7 - Scoring System Flowchart

This is the flowchart for how the scoring in the game works. Every single player starts with a score of 0, and then every single second, the 'baseScore' is added onto the current score. This loop repeats until the player dies, as the main aim of an endless runner game is to get as far as possible without dying, to get the highest score.

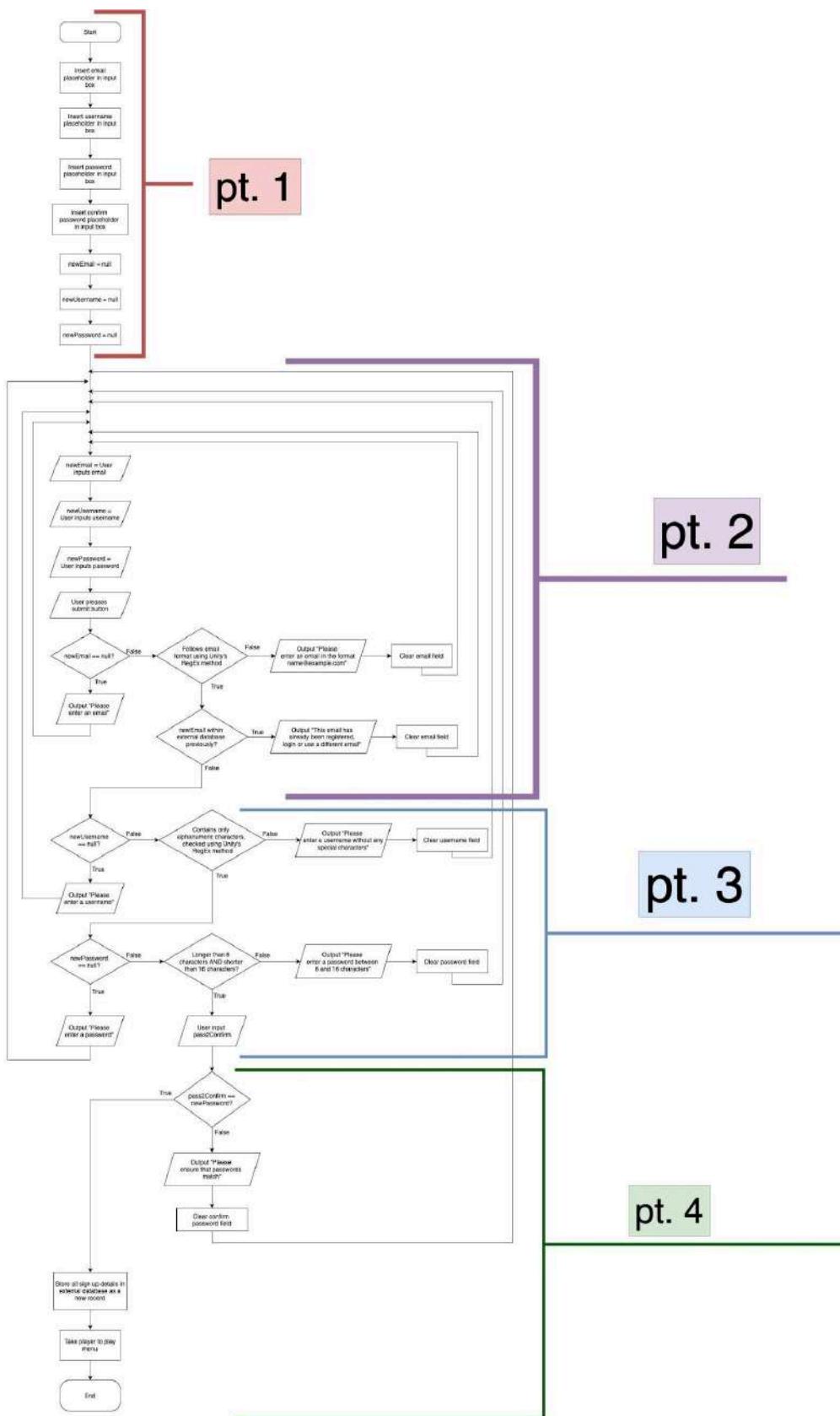
If the ability 'double score' is active, then the base score is multiplied by the score multiplier for a certain period of time. This means that the score added each time will be double what it is normally, so that the player gains more points in a shorter distance.

I have done it in terms of time because measuring distance in-game is much more difficult than calculating the time, especially because Unity has a built-in function for keeping track of time and calculating how much time has passed.



2.5.8 - Sign Up Flowchart

The entire flowchart for signing up for the game is shown below. Below this diagram, the flowchart has been split into smaller parts (as denoted on main diagram) for justifications & ease of reading.

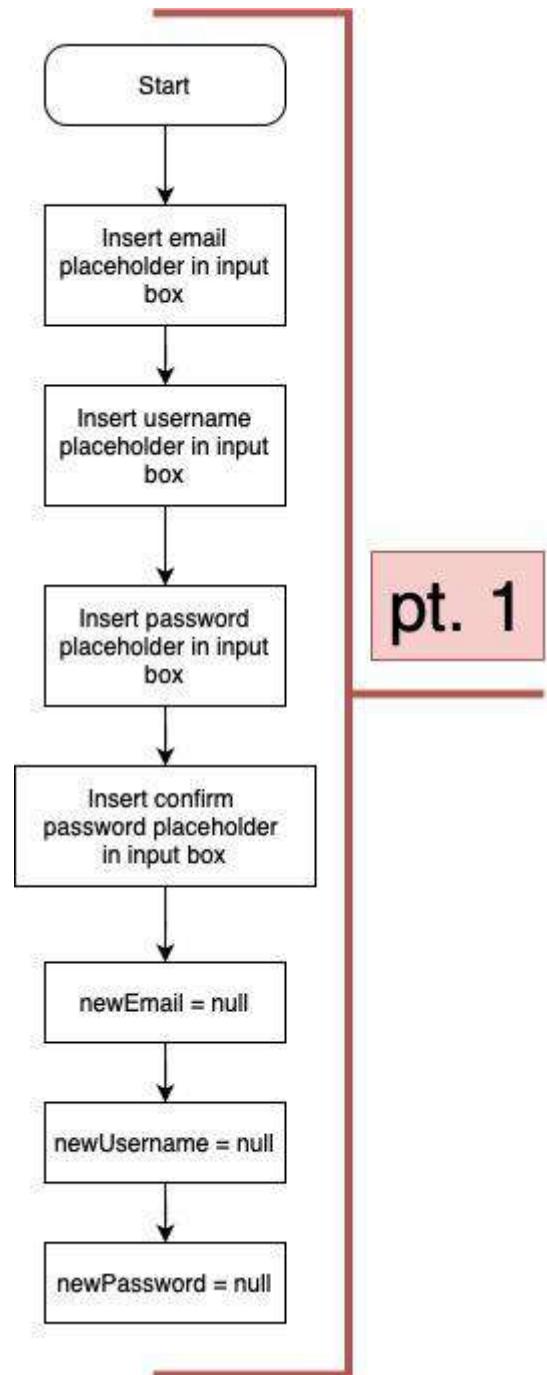


Part 1 of Sign Up

The first part of this flowchart describes the placeholders being put inside of the input boxes, and setting all of the variables that the user is about to type in as 'null' for validation reasons.

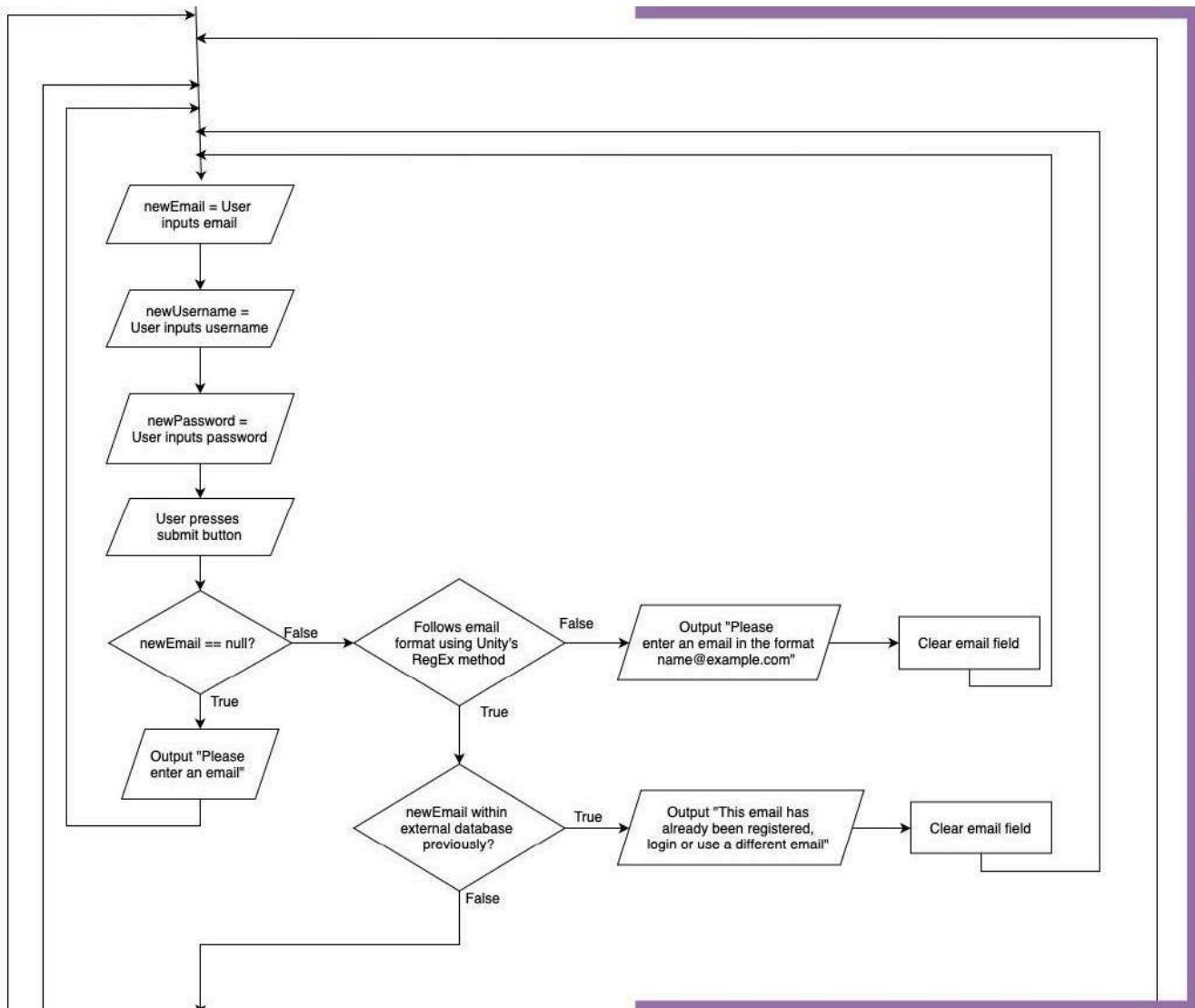
I have inserted placeholders within all of the input boxes to make it clear to the user what they should input in each box when they are signing up. This is a user-friendly approach that means that there won't be any confusion as to what each input box means.

When the user inputs something into the input box, it changes the value of newEmail or newUsername or newPassword, from null to a new value. I have decided to declare these variables as null as part of the input validation, a presence check.



Part 2 of Sign Up Flowchart

pt. 2



This is the second part of the sign up subroutine. This part begins with the user inputting their email, chosen username and password into their respective fields. Once the user presses the submit button, the validation process begins.

For the email, it first checks if the value of 'newEmail' is still null. If it is then the user is made known to input an email and then is taken through this loop until they finally enter an email address into the field. None of the other fields (username and password) are cleared when this happens.

If they have entered something into the email field, then validation begins, using Unity's RegEx (Regular Expressions) class.

- Regular Expressions: A sequence of characters (a regex pattern) that forms a search pattern for matching or finding text within strings.
- A RegEx pattern consists of characters and special symbols that define rules for matching parts of strings.

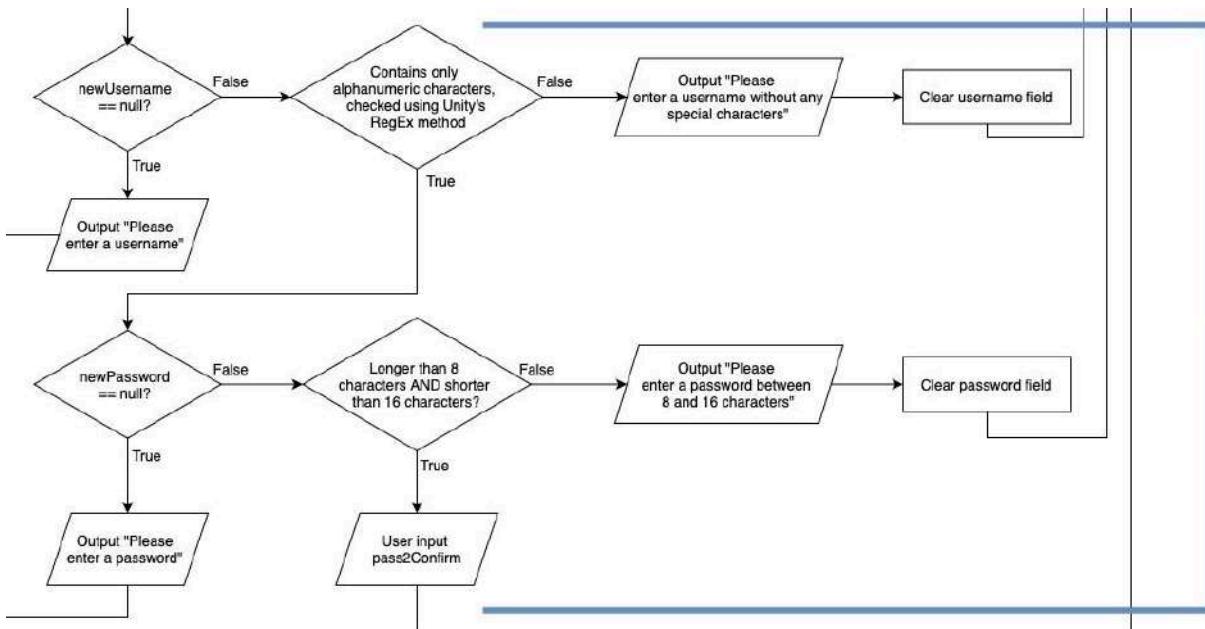
- The regex email pattern is:
`^[\w\.-]+@[a-zA-Z\d\.-]+\.[a-zA-Z]{2,}$.` where the format will follow name@example.com

If the email entered doesn't follow this format, then the user is alerted of this, the email field is cleared (username and password aren't) and the user has to re-enter an email that fits in this format. If it does, then it checks if the email has been previously registered and alerts the user if it has.

- This is to ensure that each account has a unique email address attached to it, as this is a main identifier for the user database

If all of these stages have been passed, then the validation moves onto part 3.

Part 3 of Sign Up Flowchart



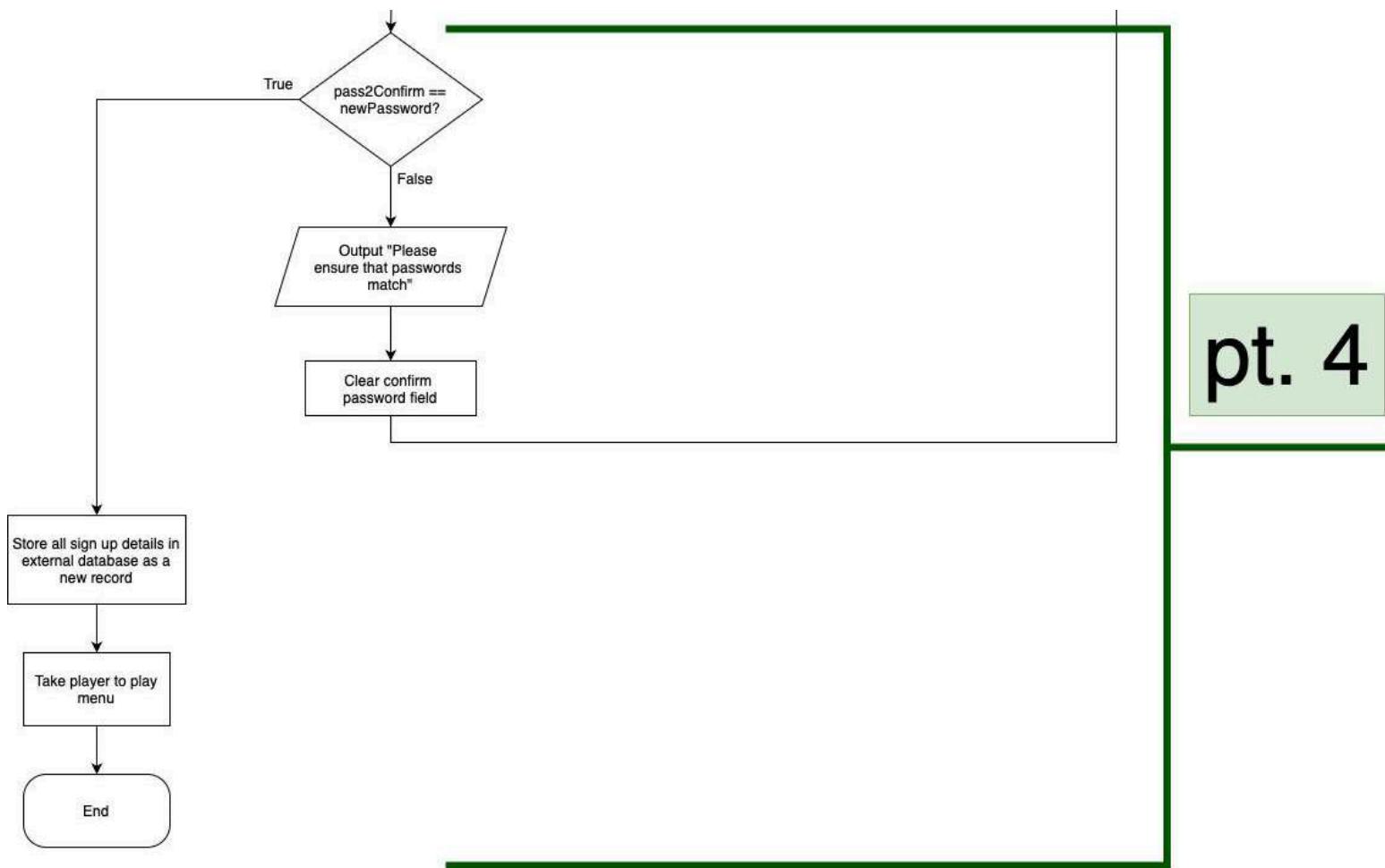
pt. 3

This is the 3rd part of the sign up process flowchart. This part begins with the validation process of the username field. The user is told to enter a username if the field is null, and if it has been filled it is then checked by Unity's RegEx method. If it contains anything other than alphanumeric characters (e.g. contains any special characters) the user is told to enter a username that doesn't contain any special characters, and the username field is cleared.

- If all of these checks are passed, then it then begins to validate the password entered

It does a presence check of the password field, and alerts the user if needed. It then checks if the password is between 8 and 16 characters, and alerts the user and clears the field if it isn't. If the password passes all of these checks, then the program then begins to check the confirm password field.

Part 4 of Sign Up Flowchart

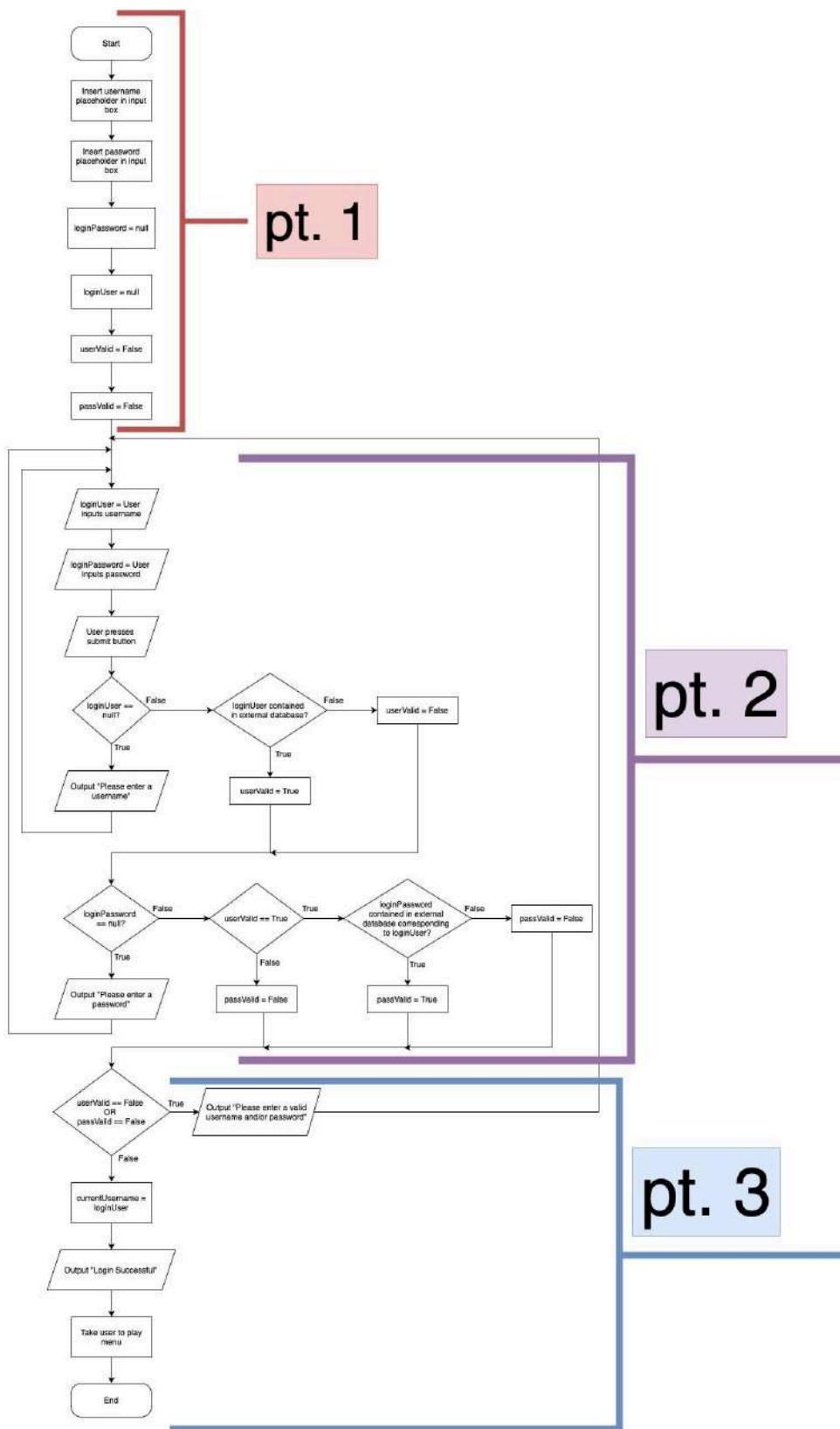


This is the last part of the sign up flowchart. In this part of the flowchart, the system has received an input of the password being entered a second time, and it checks if the confirm password input field 'pass2Confirm' contains the same value as the 'newPassword' variable. If not then the user is alerted of this and the confirm password input field is cleared, and the user can try again.

When all of these steps are completed, the player's details are stored in the external database, and the player is taken to the play menu.

2.5.9 - Log In Flowchart

This is the flowchart for logging in to AstroSprint. Below this diagram, the flowchart has been split into smaller parts (as denoted on main diagram) for justifications & ease of reading.

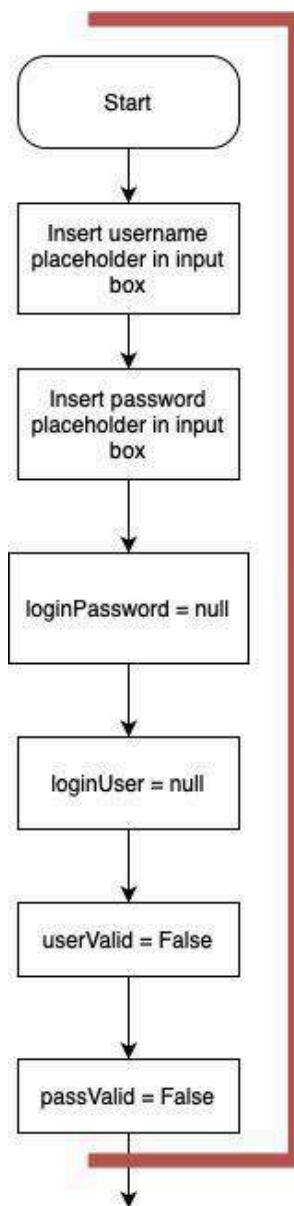


Part 1 of Login Flowchart

This is the 1st part of the login flowchart. In this part, the placeholders are firstly placed into their respective input fields. This is a user-friendly approach, which aims to make sure players aren't confused as to what details to input into the login screen to access their account. This allows a smooth and coherent flow of the game.

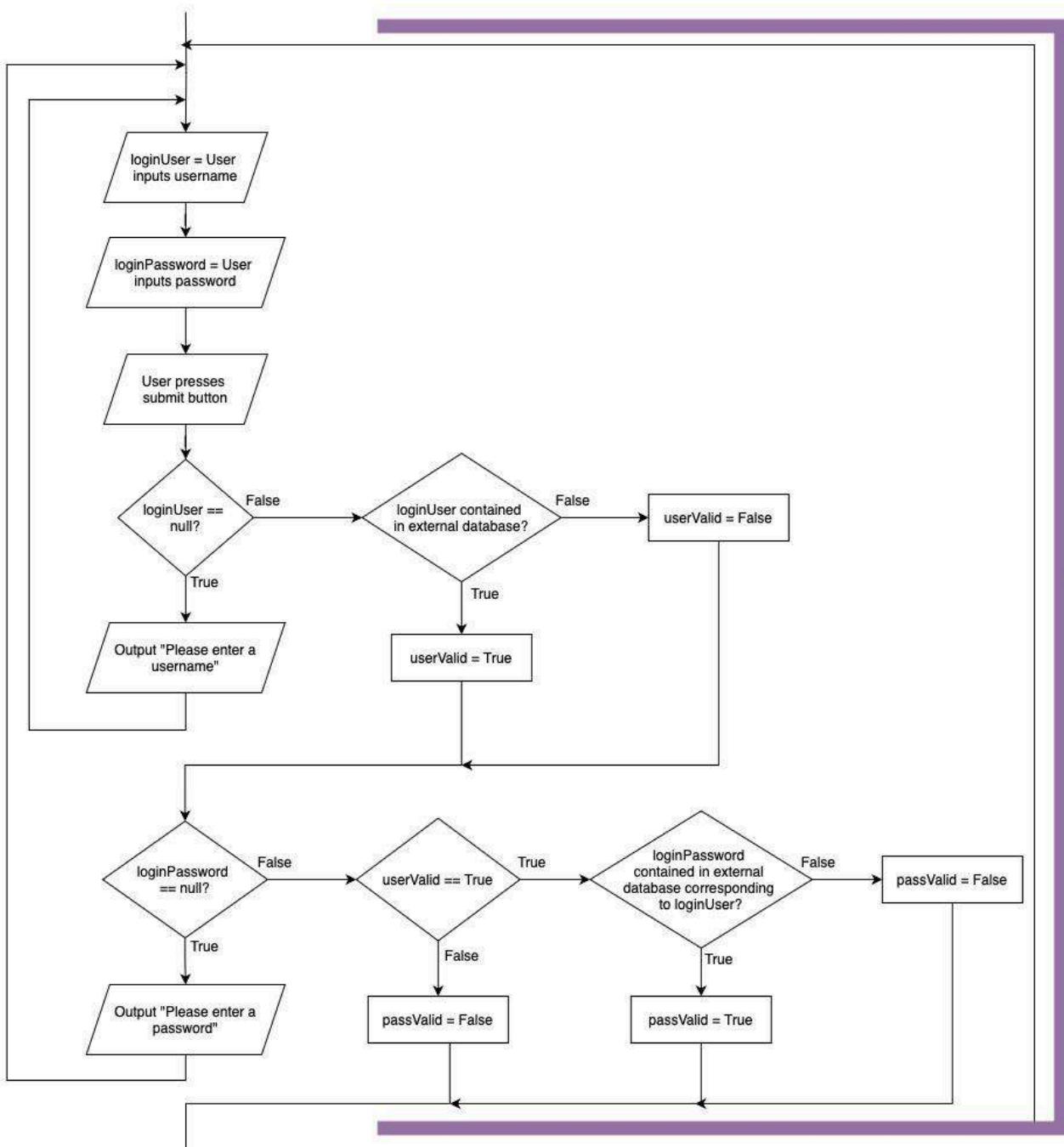
The next section focuses on initialising the variables that the input fields will eventually lead to, as 'null'. This is done for validation purposes, as if these remain null, it means that the user hasn't input anything into a certain input field.

The last section of this flowchart describes making the validation variables that check whether the username and password entered are valid, as false. This is because, before the user has entered anything, the input fields are currently 'null' which aren't valid as logins. When the user does attempt to login, the userValid and passValid must be true.



pt. 1

Part 2 of Login Flowchart



pt. 2

This is the second part of the login flowchart. This part of the flowchart begins by allowing the user to enter their username and password into their respective input fields, then press submit.

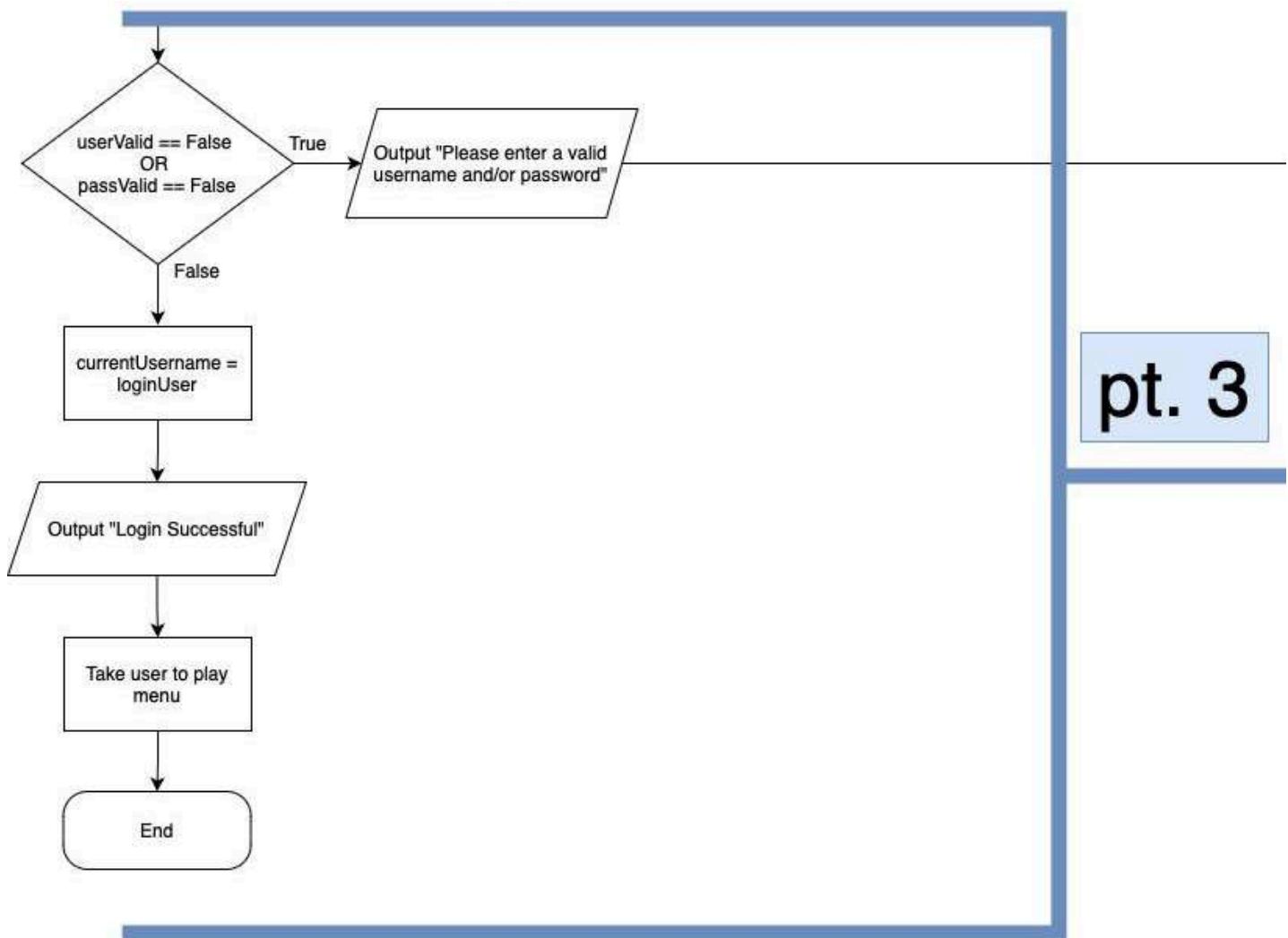
The next section of this part is validating the username that the user has just entered. If it is null, they are asked to enter a username. If they have entered a username, it is checked to see if it is in the external database. If it isn't, then the username is not valid (hence userValid = False) and if it is then the username is valid (hence userValid = True)

It then moves onto the next step of validation. In this section, the password is validated. If nothing is entered (`loginPassword = null`, as the value only changes if the user inputs something into the password input field), they are asked to enter a password. If the username isn't valid, the password is automatically invalid. This is because it is redundant to search for a password in the database where there is no corresponding username.

If the username is valid, then the '`loginPassword`' value is compared to that of the password held in the database with the username entered ('`loginUser`') to see if they are the same. If not then the password is invalid (`passValid = False`), and if it is then the password is valid (`passValid = True`).

After these checks are completed, the final validation of the login details takes place.

Part 3 of Login Flowchart



This is the final validation of the user's chosen login details. If either of the password or username are invalid, then they are forced to go through the entire login process again. The user is alerted that either their username or password is invalid, as to prevent brute force attempts of logging in.

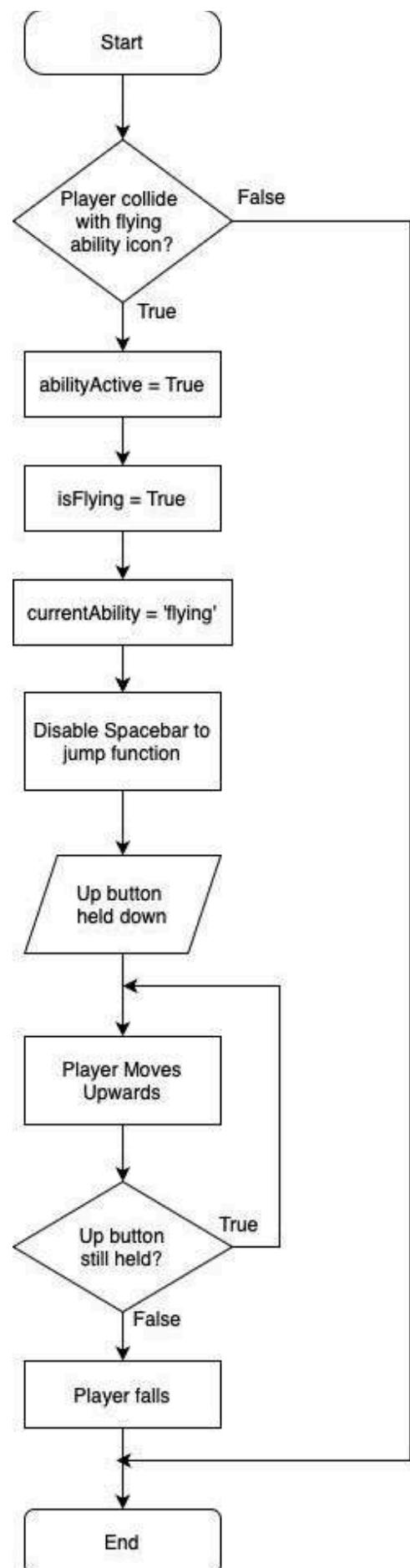
If both of these are valid, then the user is able to login, the `currentUsername` is set to the one they used to login and they are taken to the play menu to play the game.

2.5.10 - Flying (Power Up) Flowchart

This flowchart describes the process of flying when the flying ability is collected. It starts when the player collides with the flying balitty icon. After this, the abilityActive and isFlying variables are set to true. To be able to fly, the concept of jumping must be disabled. I have done this to ensure that the system doesn't get confused about what movement to give the player.

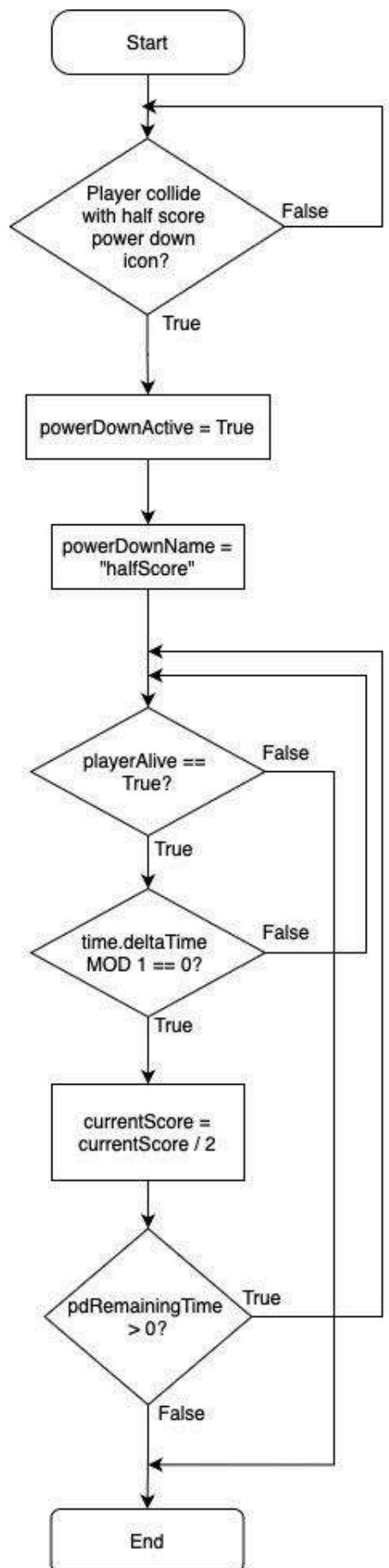
It then checks to see if the player is holding down the up button. While the player is holding it down they will move upwards. Once they let it go, the player will fall.

This functionality is carried out until the player dies or the flying ability isn't active anymore.



2.5.11 - Half Score (Power Down) Flowchart

This flowchart describes how the half score power down works. When the player collides with the power down icon for half score, as long as the player is alive, their current score will be halved every single second, until the power down runs out of time.



2.5.12 - Blurry Screen (Power Down) Flowchart

This flowchart describes how the power down 'Blurry Screen' works. When the player collides with the blurry screen power down icon, as long as the player is alive, a blurry overlay will be placed over the top of the screen. This hinders the player's ability to see during the game, making them more likely to hit an obstacle.

Once the remaining time is no longer above 0, the overlay is removed and the subroutine ends.

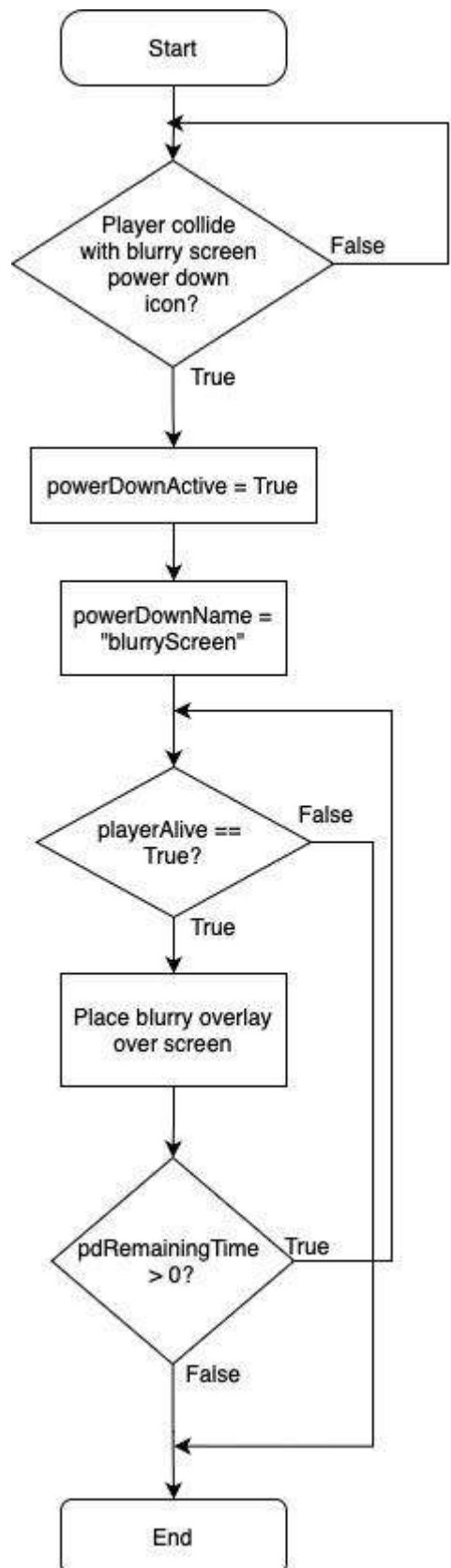
I have made sure that the loop repeats only if the player is alive because if they are dead, all effects of the game should be removed. If not, this could cause extreme difficulty when trying to navigate the menus.

2.5.12.1 - Pseudocode Representation

```
if player.CollidesWith(blurryScreenIcon) then
    powerDownActive = True
    powerDownName = "blurryScreen"
    while pdRemaining > 0
        if playerAlive == True then
            blurryOverlay.setActive()
        else
            blurryOverlay.setFalse()
        endif
    endwhile
endif
```

2.5.12.2 - Algorithmic Representation

- If the player collides with the blurry screen icon, then
 - The powerDownActive variable is set to true
 - The powerDownName is set to be the name of the power down, 'blurry screen'
 - if the player is alive, then
 - while there is still time remaining on the power down's duration
 - the blurry screen overlay is set to be visible
 - If there is no time remaining on the power down's duration the blurry overlay is hidden



2.5.13 - Darkness (Power Down) Flowchart

This flowchart describes how the power down 'Darkness' works. When the player collides with the darkness power down icon, as long as the player is alive, a dark overlay will be placed over the top of the screen. This hinders the player's ability to see during the game, making them more likely to hit an obstacle.

Once the remaining time is no longer above 0, the overlay is removed and the subroutine ends.

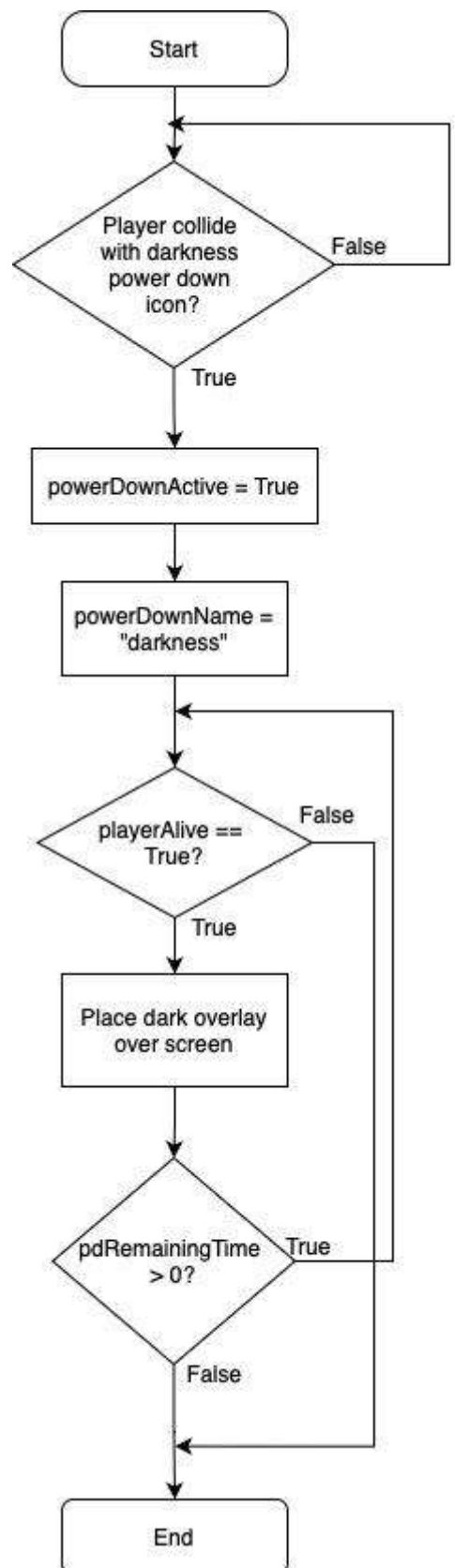
I have made sure that the loop repeats only if the player is alive because if they are dead, all effects of the game should be removed. If not, this could cause extreme difficulty when trying to navigate the menus.

2.5.12.1 - Pseudocode Representation

```
if player.CollidesWith(darknessIcon) then
    powerDownActive = True
    powerDownName = "darknessScreen"
    while pdRemaining > 0
        if playerAlive == True then
            darkOverlay.setActive()
        else
            darkOverlay.setFalse()
        endif
    endwhile
endif
```

2.5.12.2 - Algorithmic Representation

- If the player collides with the darkness icon, then
 - The powerDownActive variable is set to true
 - The powerDownName is set to be the name of the power down, 'darkness'
 - if the player is alive, then
 - while there is still time remaining on the power down's duration
 - the dark overlay is set to be visible
 - If there is no time remaining on the power down's duration the dark is hidden

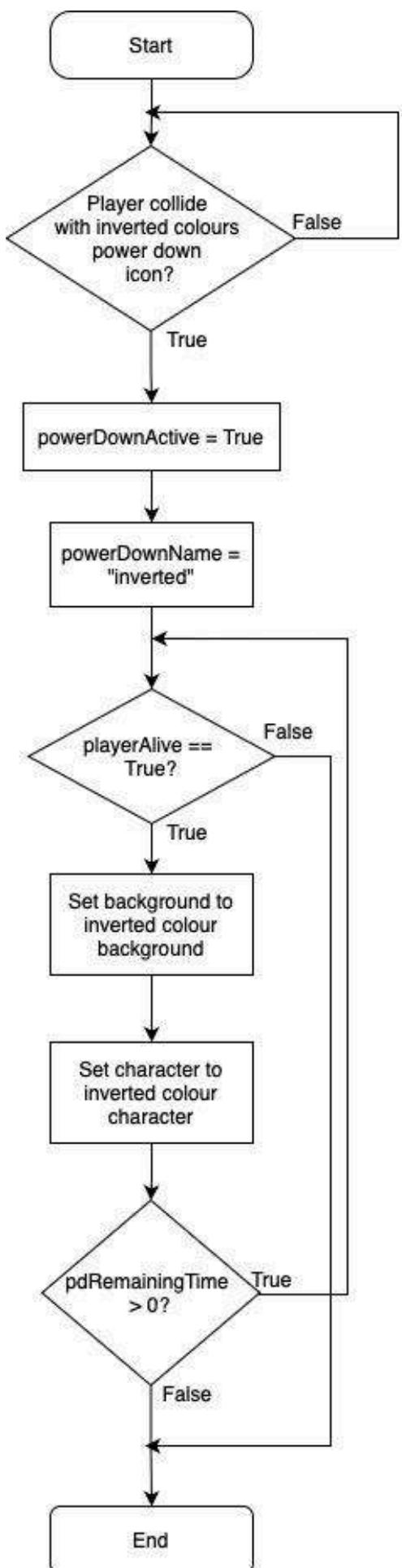


2.5.14 – Inverted Colours (Power Down) Flowchart

This flowchart describes how the power down 'Inverted Colours' works. When the player collides with the inverted colours power down icon, as long as the player is alive, the in-game colours will be replaced with their inverted versions. This surprises the player and makes them lose their focus, making them more likely to hit an obstacle.

Once the remaining time is no longer above 0, the colours return to normal and the subroutine ends.

I have made sure that the loop repeats only if the player is alive because if they are dead, all effects of the game should be removed. If not, this could cause extreme difficulty when trying to navigate the menus.



2.5.15 – Character Selection Flowchart

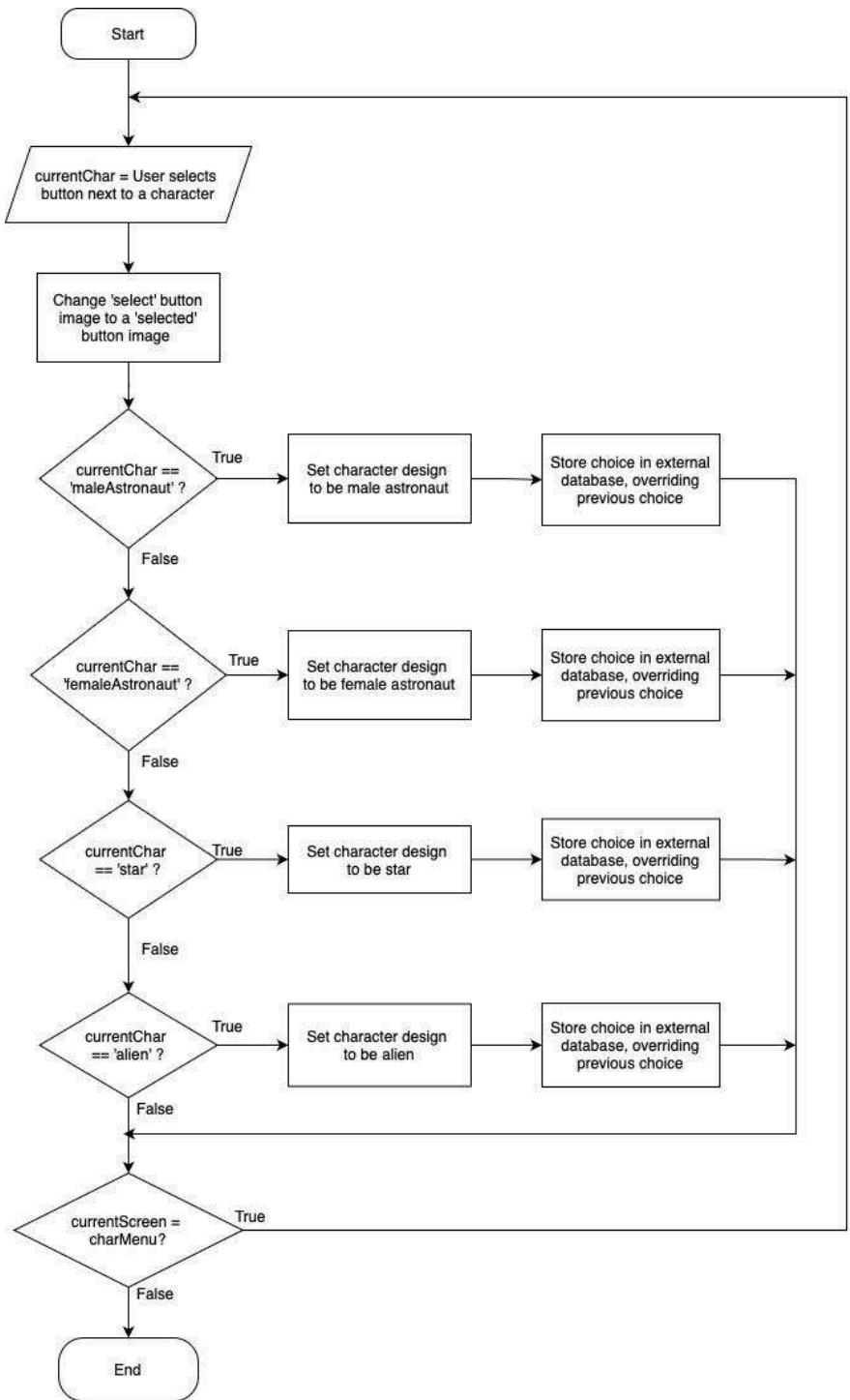
This is the flowchart for the character selection when the user is on the character menu.

This flowchart starts with user input. This is because, if the user doesn't select anything, the character doesn't change. The current character is set to 'male astronaut' by default.

After a button is clicked, it is then changed from a 'select' to a 'selected' button. This is done to show to the user that their character has been changed.

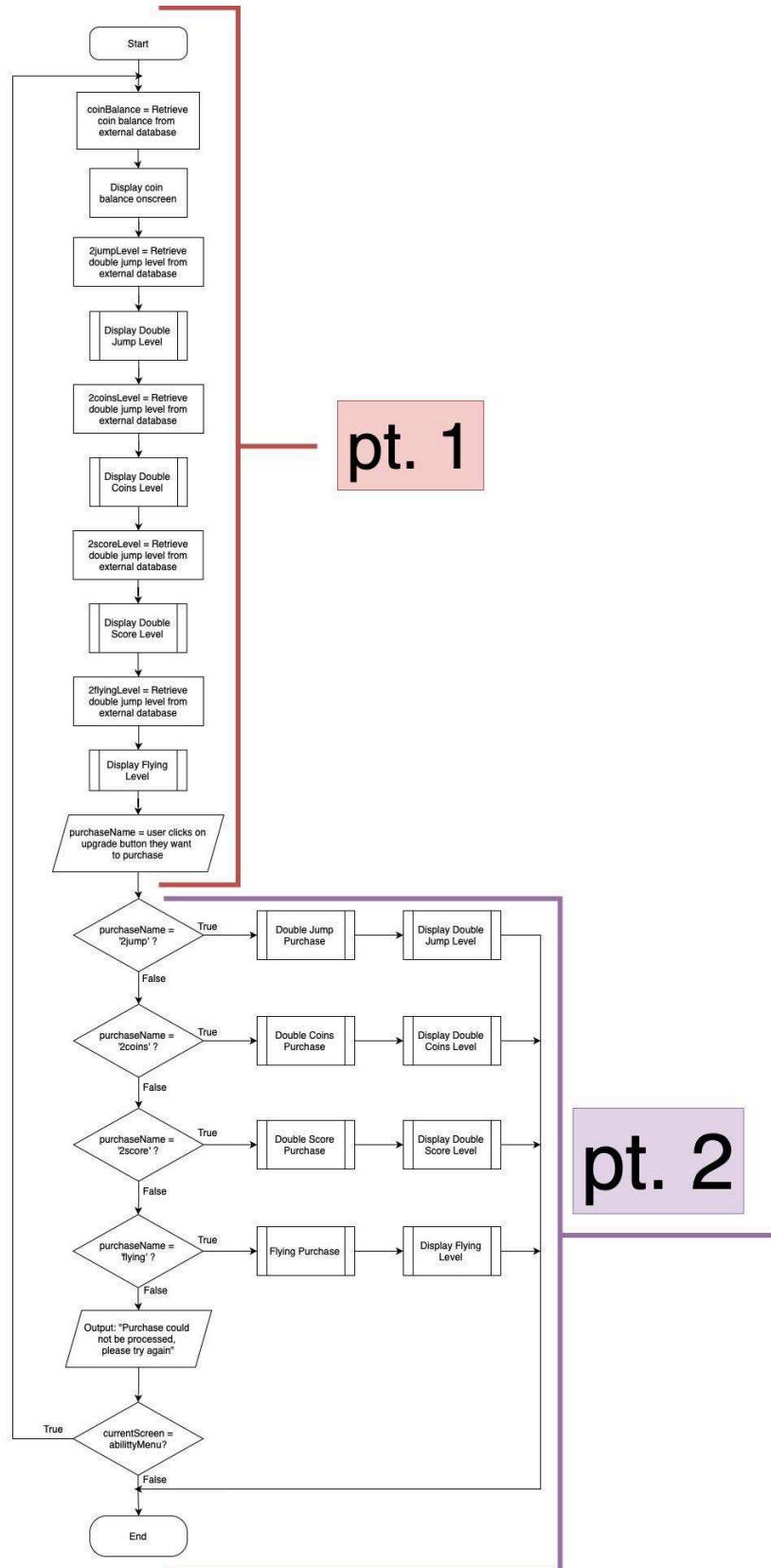
The next steps just check what character has been selected, and send the name of it to the external database. This is done so that the character choice is saved for the next time that the player logs in.

All of this is in a loop that continues as long as the user remains on the character menu. This is so that the player can change their character as many times as they want.



2.5.16 - Abilities Upgrade Flowchart

The entire flowchart for upgrading the abilities in the game is shown below. Below this diagram, the flowchart has been split into smaller parts (as denoted on main diagram) for justifications & ease of reading.

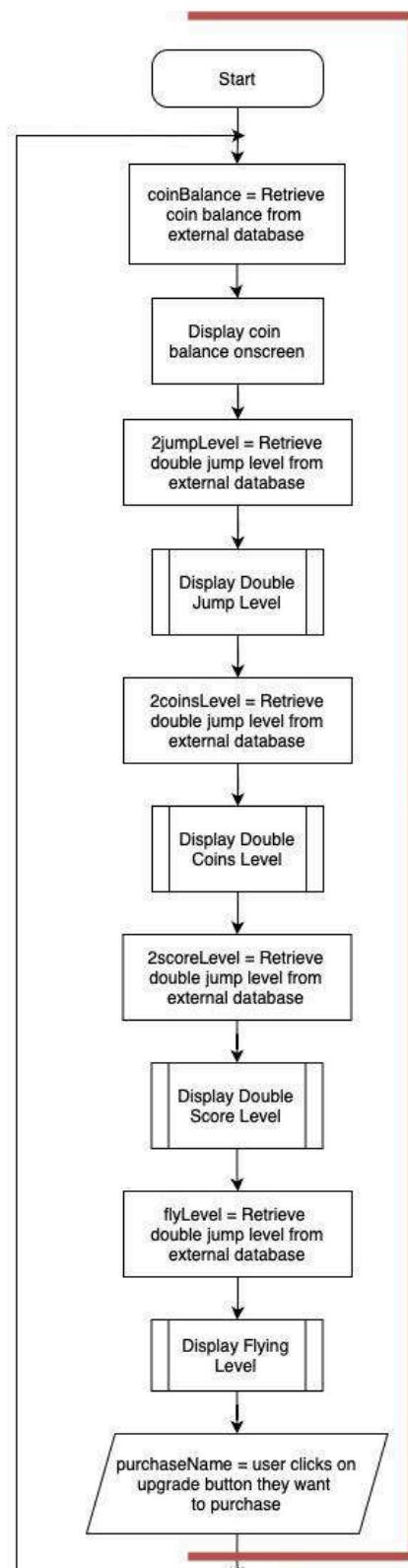


Part 1 of Abilities Upgrade Flowchart

This is the first part of the ability upgrade flowchart. The first section of this retrieves the coinBalance from the external database and displays it to the user. This is done so that the user knows what their coin balance currently is, and see what they can afford with their in-game currency.

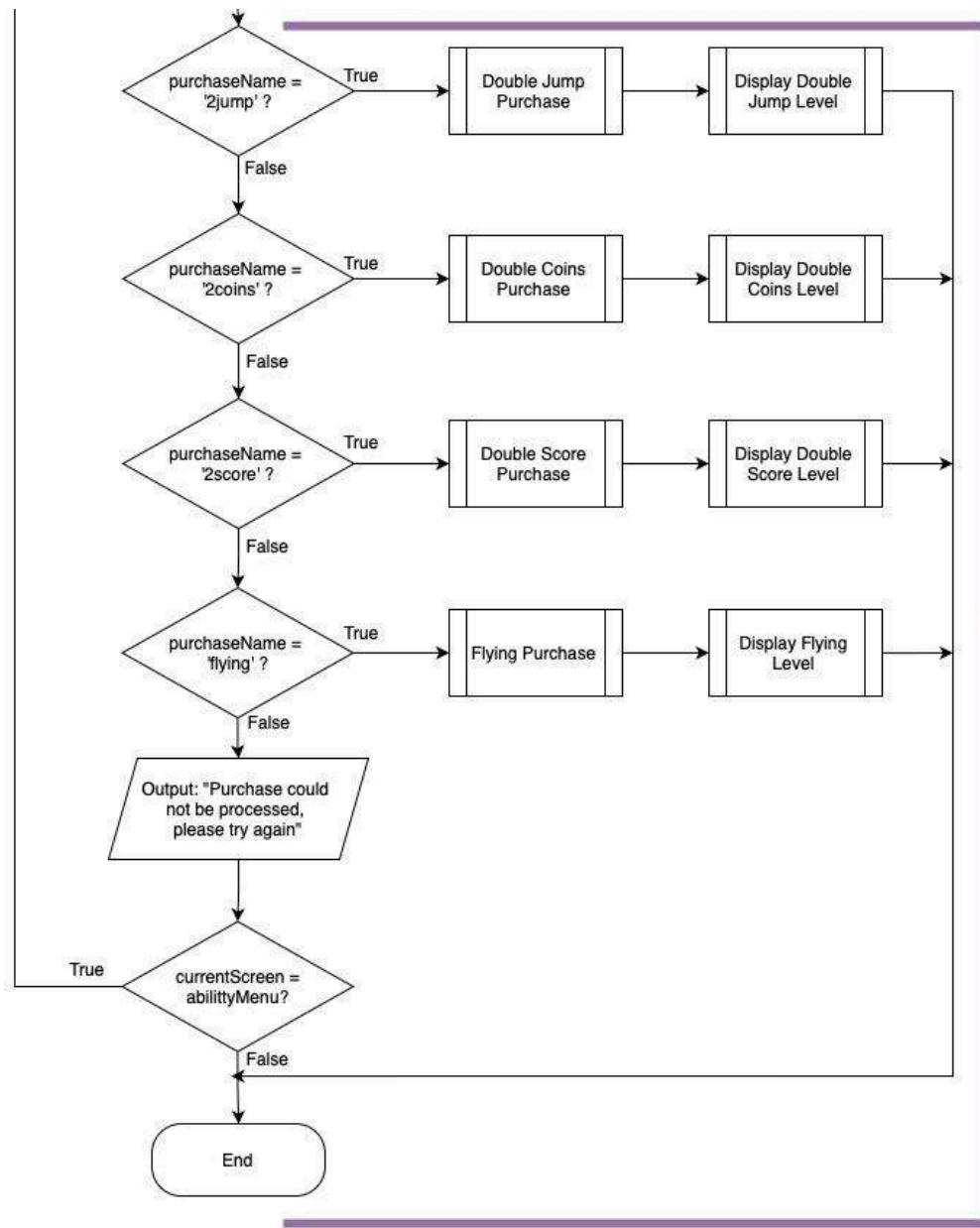
The next part of this flowchart is retrieving the various levels of all the different power ups, and using a subroutine to display this to the user. This is done so that the user can visually see how much power ups have been upgraded, and see the duration that their power up currently lasts for.

After this, the name of the purchase is declared to be the upgrade button of what they want to purchase.



pt. 1

Part 2 of Abilities Upgrade Flowchart



pt. 2

The second part of this flowchart is shown above. The majority of this part is the process of, depending on the name of the power up purchase, which power up to upgrade, and then displaying the new state of that power up. If the purchaseName doesn't match to any of the existing power ups, then a message is sent to the user. This is so that they are made aware of why their power ups may not be upgrading.

All of this is in a loop that continues as long as the user remains on the abilities menu. This is so that the player can upgrade their abilities as many times as they want.

2.5.17 - Displaying Ability Levels Flowcharts

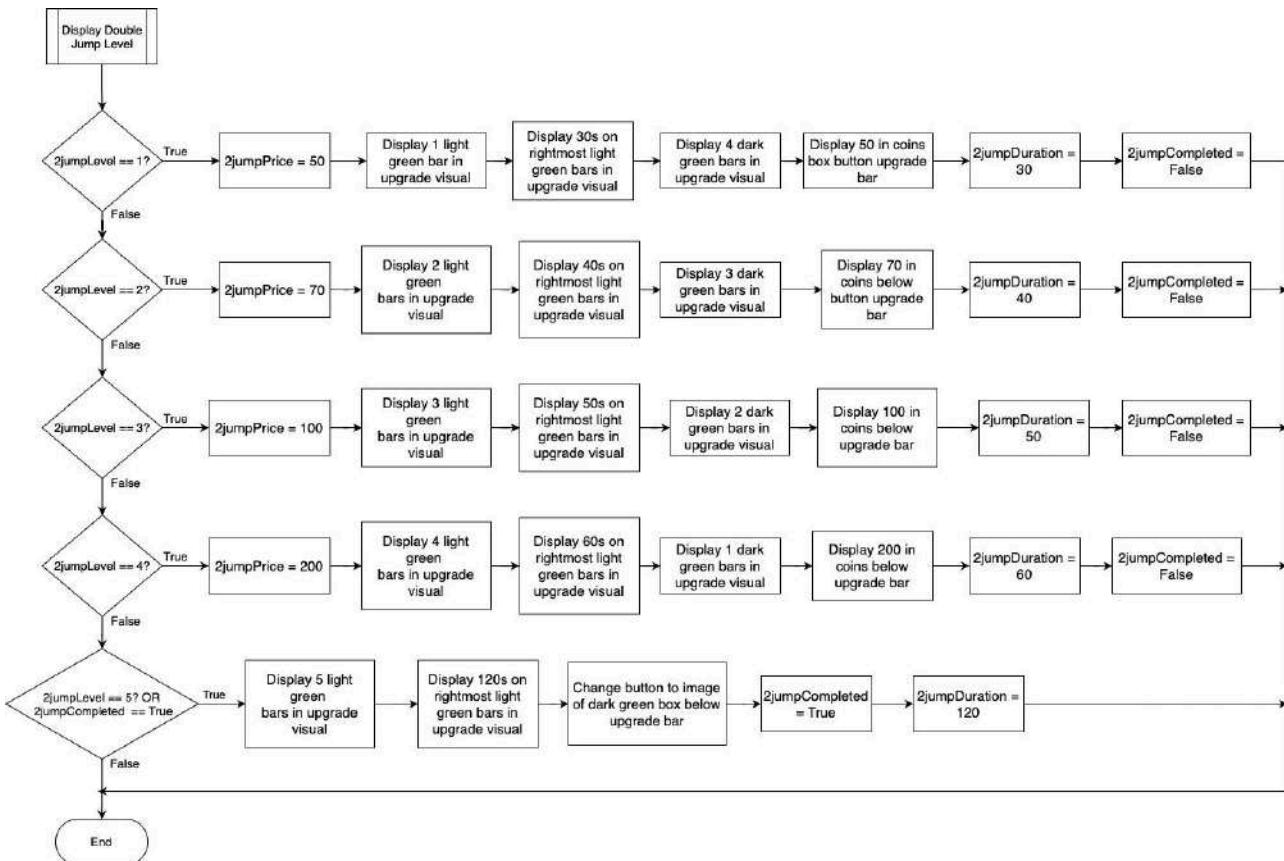
Below are 4 flowcharts for the displaying of ability levels. This is an identical process for all of the different power ups, and so have the same flowchart, apart from differences in variable names.

It first begins by checking what the current level of the power up is. It then sets the current price of the power up, as it varies depending on the current level that the power up is currently upgraded to. The pricing changes depending what level the user is currently on because the increase in duration after each level increases time, and to create a sense of challenge.

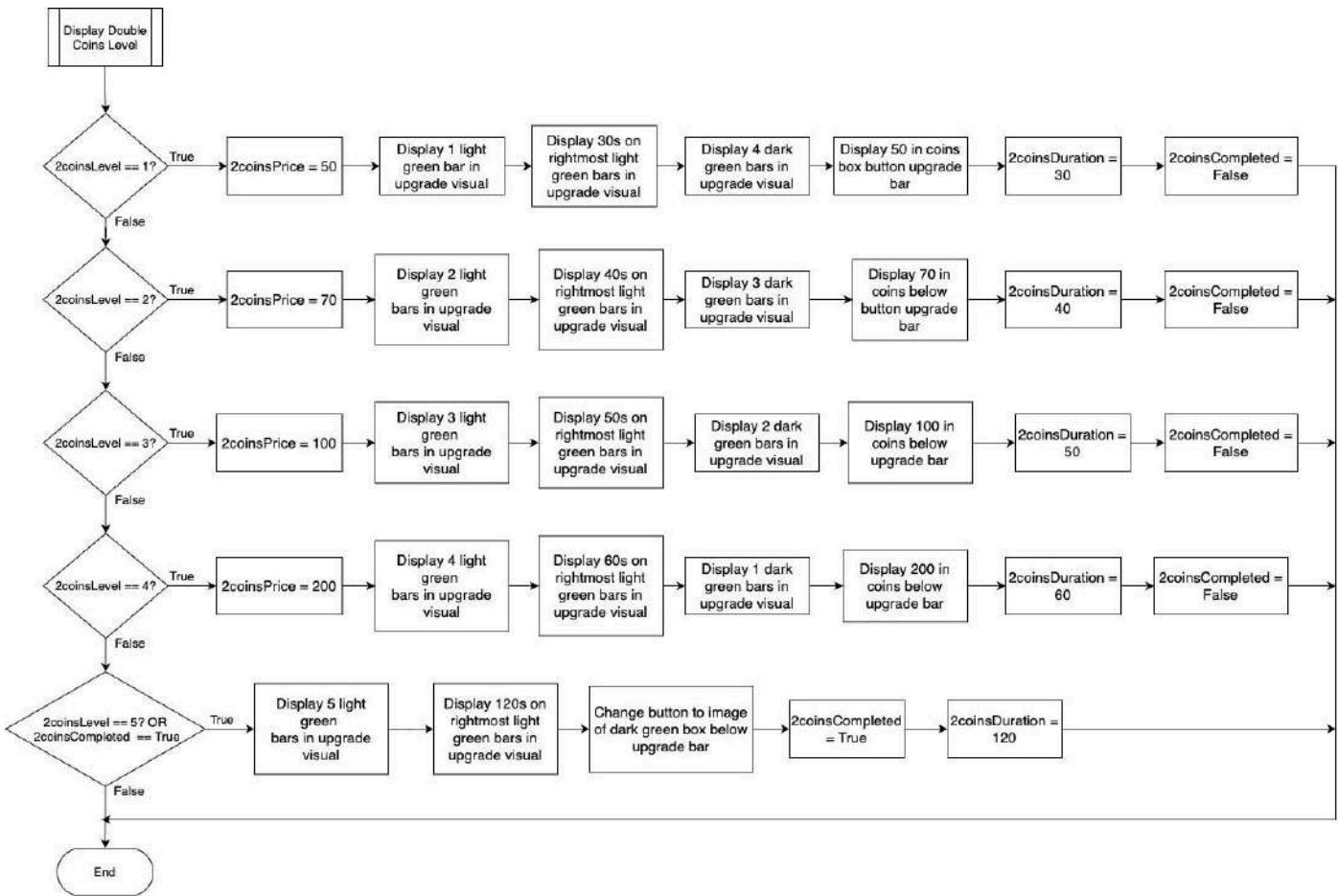
The level that the power up is currently at is described as a light green bar, in which the rightmost green bar shows the actual numerical value of time that the power up lasts for.

If the power up is at level 5 (which is when it is completed) all of the bars are light green and what used to be a button showing the current price of the next upgrade, becomes a dark green box. This symbolises the idea that the power up can no longer be upgraded.

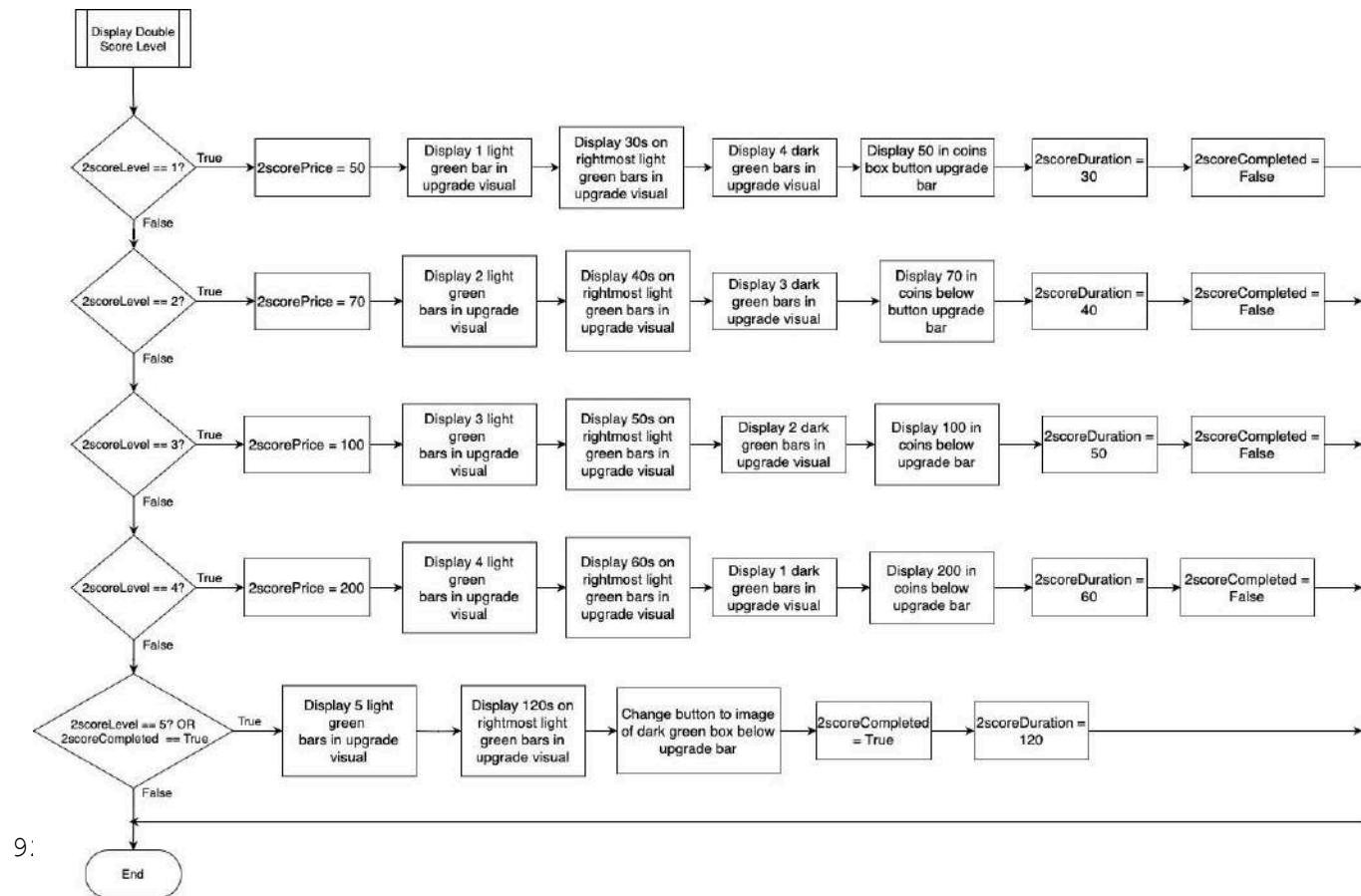
Displaying Double Jump Level



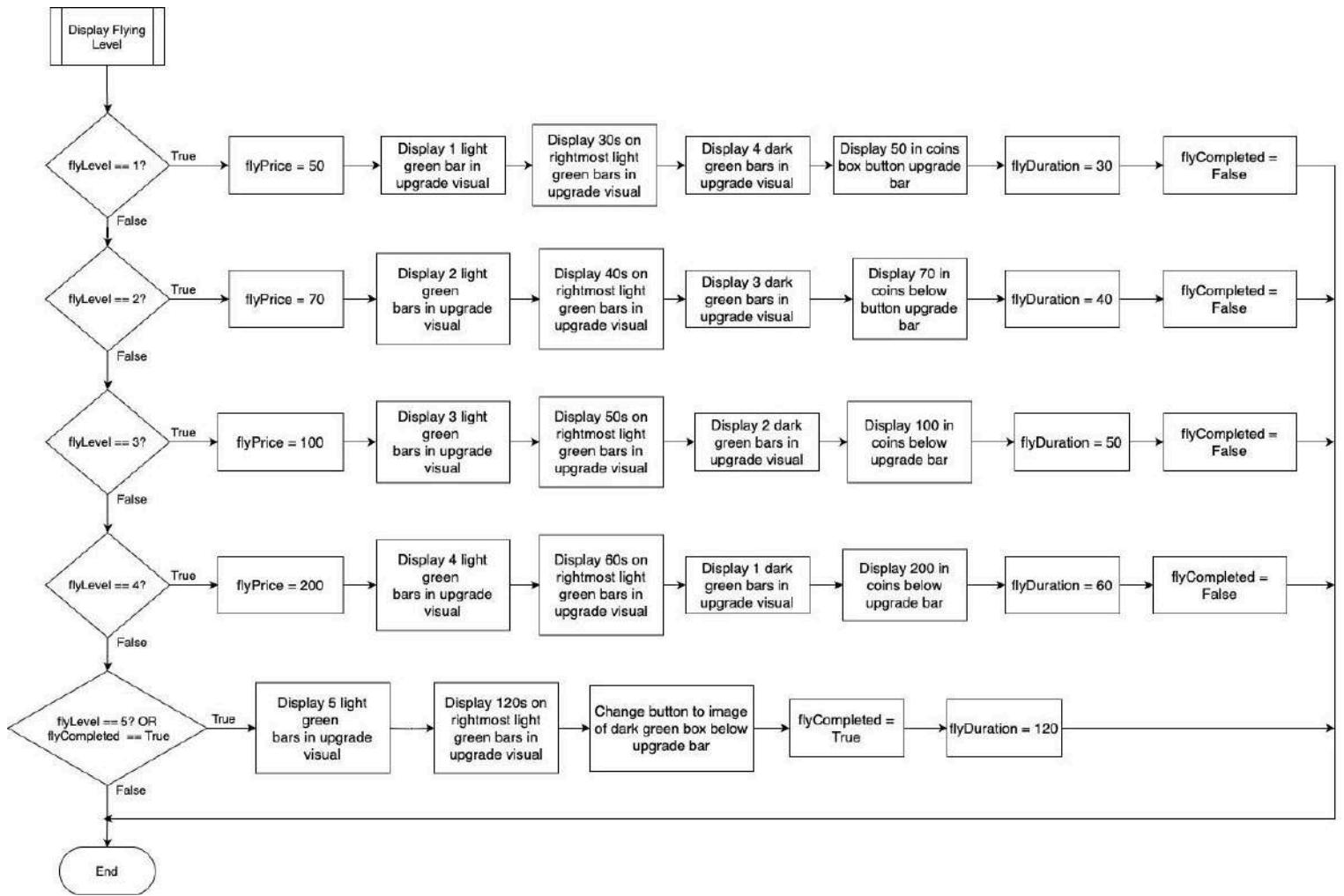
Displaying Double Coins Level



Displaying Double Score Level



Displaying Flying Level



2.5.18 – Purchasing Ability Upgrade Subroutine

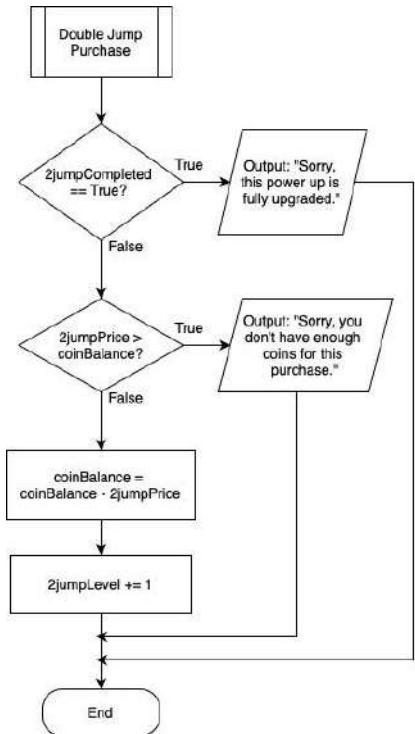
Flowcharts

The 4 flowcharts (1 to the right and 3 below) are the flowcharts for how purchasing ability upgrades work. The one to the right is the double jump, and the other 3 are named in the table below.

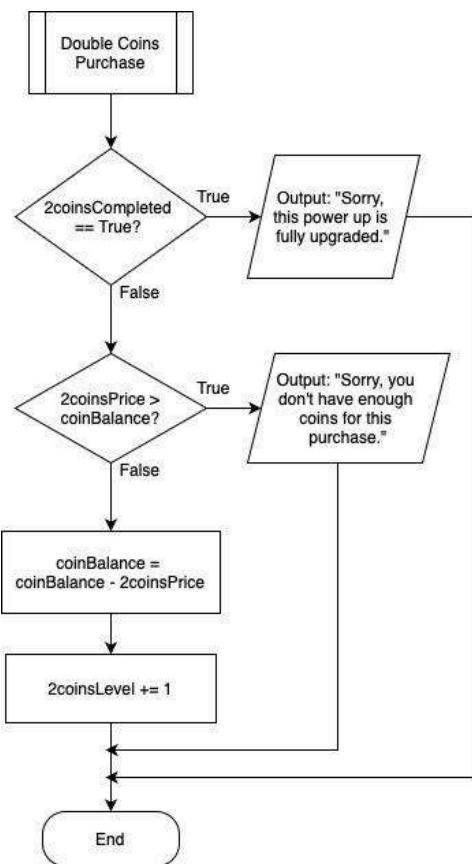
At first, it checks to see if the power up is fully upgraded. If it is, then it alerts the user, and doesn't allow any purchase to happen. This is done to avoid the user losing coins in a false purchase and not gaining anything from it.

If the power up is not completed, then if the price of the upgrade is more than the balance the player has, the user is alerted of this, and is not allowed to make the purchase. If they do have enough coins for the purchase, then the price is taken from the user's coin balance, and they gain 1 upgrade level.

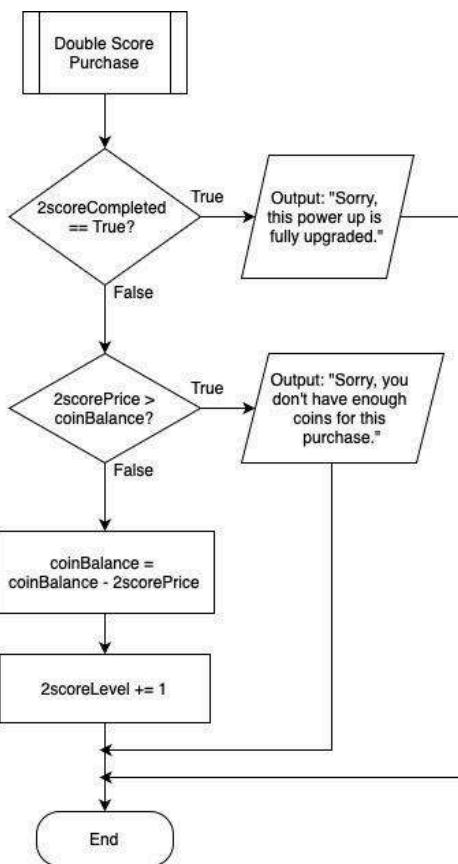
This upgrade level is reflected in the displaying ability level subroutine, described in the flowchart above.



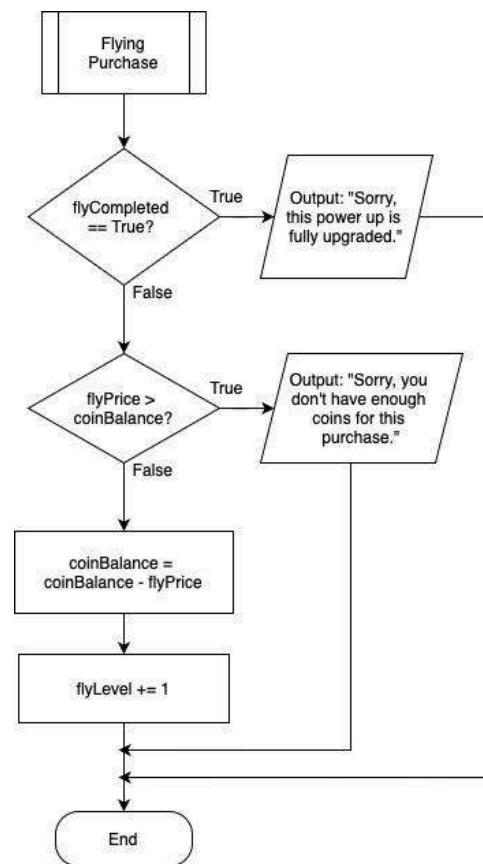
Double Coins Flowchart



Double Score Flowchart



Flying Flowchart



2.5.19 - Generating Power Ups Flowchart

This is the flowchart that describes how power ups will be generated in-game.

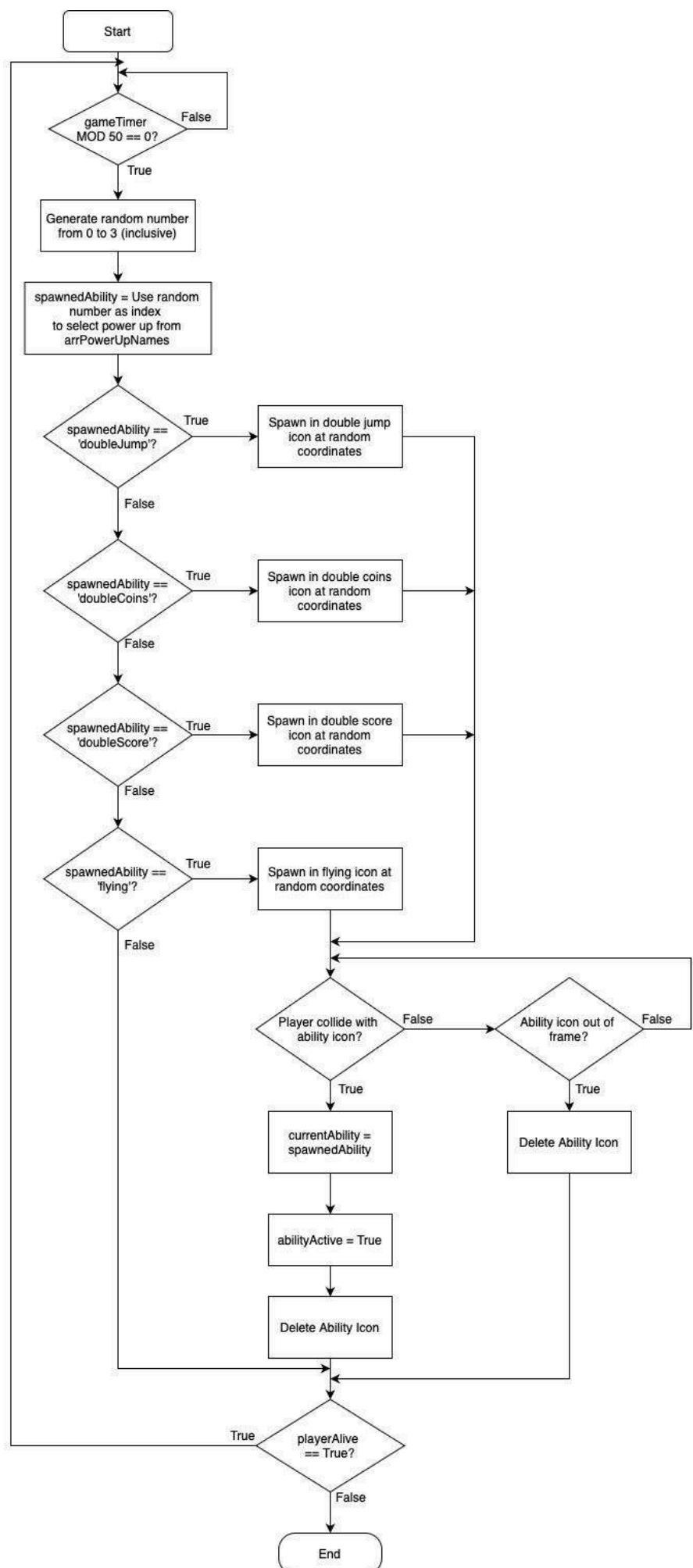
They will be generated every 50 seconds, this is to prevent the overlap of power ups (2 power ups being activated at the same time)

The power up to be generated will be randomly picked, through the index of its new in an array holding all the names of the power ups.

Once a power up is picked, the corresponding icon will be spawned in-game at a random coordinate.

If the player collides with the icon, they can collect it and use it. If the player doesn't collide with the icon, it stays spawned in the game until it is out of frame, where it is then deleted.

This loop repeats as long as the player is alive.



2.5.20 – Generating Power Downs Flowchart

This is the flowchart that describes how power downs will be generated in-game.

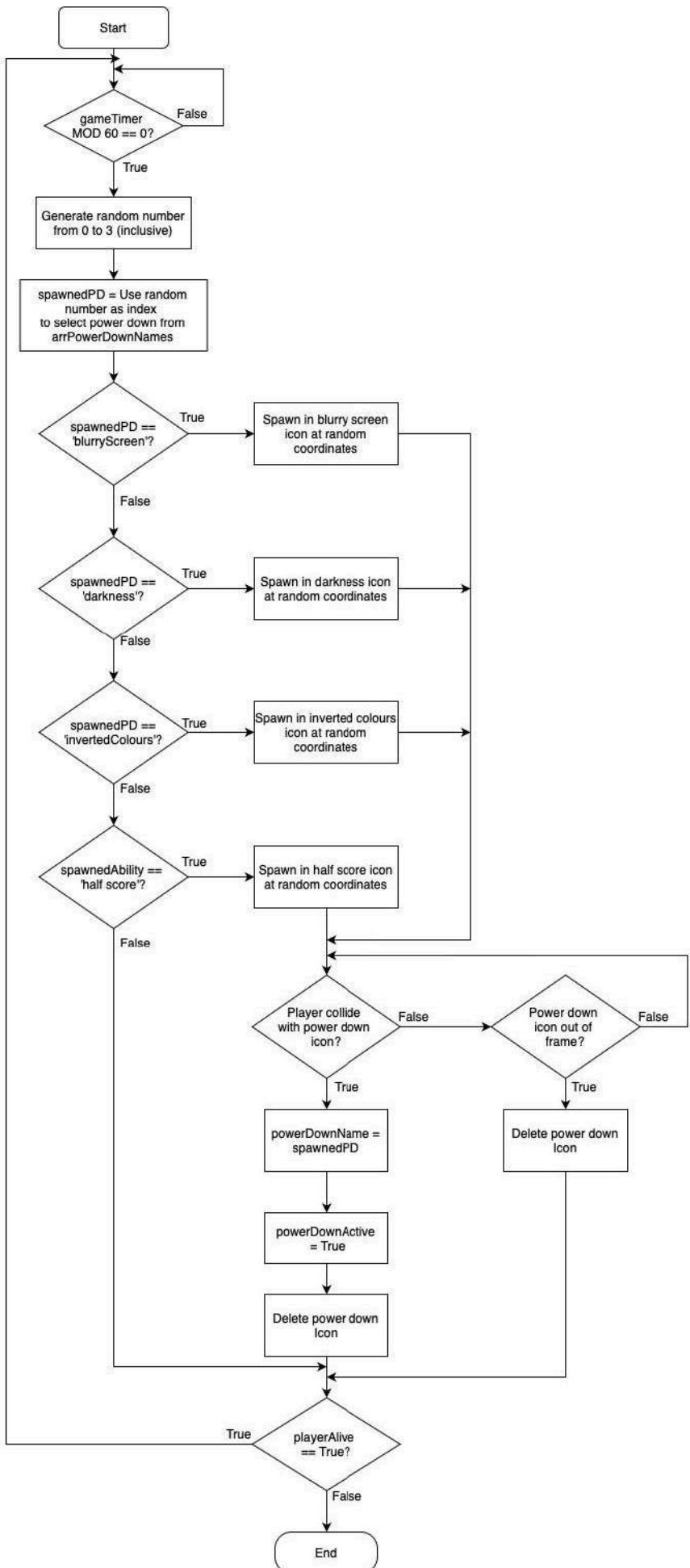
They will be generated every 60 seconds, this is to prevent the overlap of power downs (2 power downs being activated at the same time)

The power down to be generated will be randomly picked, through the index of its new in an array holding all the names of the different power downs.

Once a power down is picked, the corresponding icon will be spawned in-game at a random coordinate.

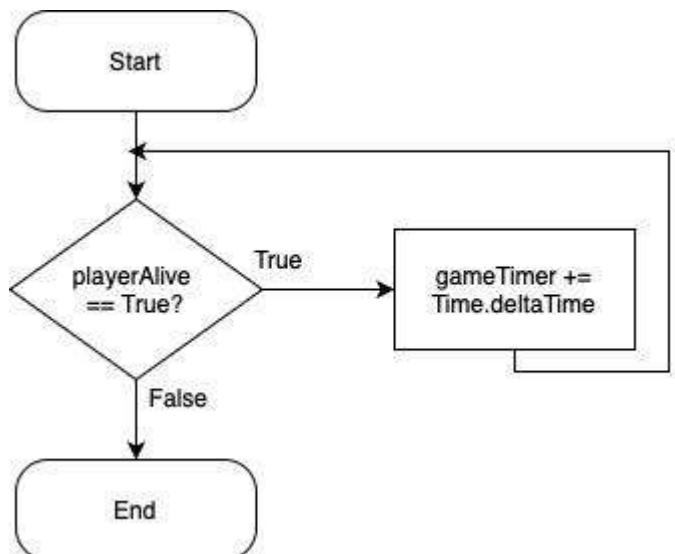
If the player collides with the icon, they will be immediately affected by it. If the player doesn't collide with the icon, it stays spawned in the game until it is out of frame, where it is then deleted.

This loop repeats as long as the player is alive.



2.5.21 - Game Timer Flowchart

This is the flowchart showing how the game timer is updated in game. The 'gameTimer' variable is made to keep track of how long the game has been running. The function 'Time.deltaTime' measures the interval between the current frame and the last one. By adding this value (as long as the player is alive) to the 'gameTimer' variable, the amount of time that the game has been running is calculated.



2.5.21.1 - Pseudocode Representation

```
gameTimer = 0
if playerAlive == True then
    gameTimer = gameTimer + Time.deltaTime()
endif
```

2.5.21.2 - Algorithmic Representation

- If the playerAlive variable has a value of 'True' then:
 - The game timer is increased by the value produced by the function Time.deltaTime
- If the playerAlive variable has a value that is not 'True' then:
 - The subroutine ends.

2.6 - Key Variables

2.6.1 - Sign Up Key Variables

Name	Data Type	Scope	Private/Public	Purpose
newUsername	string	Local to the 'SignUp' script	Private, dynamic	The user's choice of username
newPassword	string	Local to the 'SignUp' script	Private, dynamic	The first instance of the user's choice of password
newEmail	string	Local to the 'Sign Up' script	Private, dynamic	The email of the user signing up
pass2Confirm	string	Local to the 'Sign Up' script	Private, dynamic	User re-types password to match, used to confirm password
passMatch	Boolean	Local to the 'passConfirmation' subroutine	Private, dynamic	True/False for if passwords match, for validation
uniqueUser	Boolean	Local to the 'checkNewUser' subroutine	Private, dynamic	True/False for if username entered is unique
uniqueEmail	Boolean	Local to the 'checkNewEmail' subroutine	Private, dynamic	True/False for if username entered is unique
usernamePH	string	Attribute of the Sign Up Subroutine	Private, static	A placeholder for the username input box as an indicator for the user to type in their chosen username in the box
passwordPH	string	Attribute of the Sign Up Subroutine	Private, static	A placeholder for the password input box as an

				indicator for the user to type in their chosen password in the box
emailPH	string	Attribute of the Sign Up Subroutine	Private, static	A placeholder for the email input box as an indicator for the user to type in their email in the box
confirmPassPH	string	Attribute of the Sign Up Subroutine	Private, static	A placeholder for the confirm password input box as an indicator for the user to retype their password in the box

2.6.2 - Login Key Variables

Name	Data Type	Scope	Private/Public	Purpose
loginUser	string	Local to the 'Login' subroutine	Private, dynamic	the username the user is trying to use to login
loginPassword	string	Local to the 'Login' subroutine	Private, dynamic	the password the user is trying to use to login
currentUsername	string	Attribute of the User class	Private, static	The username of the user that is logged in
userValid	Boolean	Local to the 'Login' subroutine	Private, dynamic	True/False for if username entered is valid (contained in external database)

passValid	Boolean	Local to the 'Login' subroutine	Private, dynamic	True/False for if username entered is valid (contained in external database for corresponding username)
-----------	---------	---------------------------------	------------------	---

2.6.3 - Play Menu Key Variables

Name	Data Type	Scope	Private/Public	Purpose
playPrevScreen	string	Attribute of the Play Menu module	Public, dynamic	Holds the name of the previous screen, for use with the back button
currentScreen	string	Entire Program	Public, dynamic	Holds the name of the current screen being shown to the user

2.6.4 - Tutorial Key Variables

Name	Data Type	Scope	Private/Public	Purpose
tutPrevScreen	string	Attribute of the Tutorial module	Public, dynamic	Holds the name of the previous screen, for use with the back button

2.6.5 - Abilities Menu Key Variables

Name	Data Type	Scope	Private/Public	Purpose
coinBalance	integer	Attribute of the Player class	Private, dynamic	The balance of coins for the current player
abPrevScreen	string	Attribute of the Abilities Menu module	Public, dynamic	The name of the previous screen, for use

				with the back button
2jumpLevel	integer	Attribute of the Abilities Menu module	Private, dynamic	The level of upgrade that the double jump ability has reached
flyLevel	integer	Attribute of the Abilities Menu module	Private, dynamic	The level of upgrade that the flying ability has reached
2coinsLevel	integer	Attribute of the Abilities Menu module	Private, dynamic	The level of upgrade that the double coins ability has reached
2scoreLevel	integer	Attribute of the Abilities Menu module	Private, dynamic	The level of upgrade that the double score ability has reached
2jumpCompleted	boolean	Attribute of the Abilities Menu module	Private, dynamic	Whether or not the double jump ability has been fully upgraded
2jumpPrice	integer	Attribute of the Abilities Menu module	Private, dynamic	The amount it currently costs to upgrade the double jump ability
2coinsPrice	integer	Attribute of the Abilities Menu module	Private, dynamic	The amount it currently costs to upgrade the double coins ability
2scorePrice	integer	Attribute of the Abilities Menu module	Private, dynamic	The amount it currently costs to upgrade the double score ability
flyPrice	integer	Attribute of the Abilities Menu module	Private, dynamic	The amount it currently costs to upgrade the flying ability
flyCompleted	boolean	Attribute of	Private,	Whether or not

		the Abilities Menu module	dynamic	the flying ability has been fully upgraded
2coinsCompleted	boolean	Attribute of the Abilities Menu module	Private, dynamic	Whether or not the double coins ability has been fully upgraded
2scoreCompleted	boolean	Attribute of the Abilities Menu module	Private, dynamic	Whether or not the double score ability has been fully upgraded
2jumpDuration	integer	Attribute of the Abilities Menu module	Public, dynamic	Duration of the double jump ability
flyDuration	integer	Attribute of the Abilities Menu module	Public, dynamic	Duration of the flying ability
2coinsDuration	integer	Attribute of the Abilities Menu module	Public, dynamic	Duration of the double coins ability
2scoreDuration	integer	Attribute of the Abilities Menu module	Public, dynamic	Duration of the double score ability
purchaseName	string	Attribute of the Abilities Menu module	Public, dynamic	The name of the ability upgrade that the user has just purchased

2.6.6 - Character Menu Key Variables

Name	Data Type	Scope	Private/Public	Purpose
currentChar	string	Attribute of the User class	Public, dynamic	The character currently selected by the player
charPrevScreen	string	Attribute of the Character Menu module	Public, dynamic	The previous screen to the character menu, for using the back button

2.6.7 - Pause Menu Key Variables

Name	Data Type	Scope	Private/Public	Purpose
pausePrevScreen	string	Attribute of the Pause Menu module	Public, dynamic	The previous screen to the pause menu, for using the back button
pauseScore	integer	Attribute of the Pause Menu module	Public, dynamic	The score of the player when the game is paused
pauseAbTime	float	Attribute of Pause Menu Module	Public, dynamic	The time remaining on any active power ups
pausePowerUp	String	Attribute of Pause Menu Module	Public, dynamic	The power up (if any) when the player has paused
pauseObstCount	integer	Attribute of Pause Menu Module	Public, dynamic	The amount of obstacles in frame when the game is paused
pausePDTIME	float	Attribute of Pause Menu Module	Public, dynamic	the amount of time left on a power down
pausePowerDown	string	Attribute of Pause Menu Module	Public, dynamic	the name of the power down (if any) applied to a player

2.6.8 - Easy Mode Key Variables

Name	Data Type	Scope	Private/Public	Purpose
currentScore	float	Attribute of the Easy Mode module	Public, dynamic	The current score of the player
gameMode	string	Attribute of Easy Mode module	Public, dynamic	The current game mode act
obstMaxHeight	float	Attribute of	Public,	The maximum

		Easy Mode Module	dynamic	y-value that an obstacle can be generated at.
playerMove	Boolean	Attribute of Easy Mode module	public, dynamic	Whether the player is currently able to move or not
scoreMultiplier	integer	Attribute of the Easy Mode module	Public, dynamic	The current multiplier of the score
baseScore	integer	Attribute of the Easy Mode module	Private, static	The base amount that the score increases by per second
coinsCollected	integer	Attribute of the Easy Mode module	Public, dynamic	The amount of coins collected within that 1 game
abilityActive	boolean	Attribute of the Easy Mode module	Private, dynamic	Whether or not an ability is active
currentAbility	string	Attribute of the Easy Mode module	Private, dynamic	Name of the ability active (if any)
abRemainingTime	float	Attribute of the Easy Mode module	Private, dynamic	Remaining time of the ability (if any)
playerAlive	boolean	Attribute of the Easy Mode module	Public, dynamic	Whether or not the player is alive
isFlying	boolean	Attribute of the Easy Mode module	Public, dynamic	Whether or not the player is flying
isPaused	boolean	Attribute of the Easy Mode module	Public, dynamic	Whether or not the game is paused
emGameAcceleration	float	Attribute of the Easy Mode module	Private, static	The rate at which the speed of the game increases in easy mode
emStartRate	float	Attribute of	Private,	The beginning

		the Easy Mode module	static	speed of the game
jumpInterval	float	Attribute of the Easy Mode module	Public, static	The amount that the player will jump per second (increase in y-coordinate per second of a jump)
totalJumpLength	float	Attribute of Easy Mode module	Public, static	The amount of time that a jump will last
currentJumpLength	float	Attribute of Easy Mode module	private, static	The amount of time that has passed during the jump
obstacleCount	Integer	Attribute of Easy Mode Module	Public, dynamic	The number of obstacles active
gameTimer	float	Attribute of Easy Mode Module	Public, dynamic	The amount of time the game has been running

2.6.9 - Hard Mode Key Variables

Name	Data Type	Scope	Private/Public	Purpose
currentScore	float	Attribute of the Hard Mode module	Public, dynamic	The current score of the player
scoreMultiplier	integer	Attribute of the Hard Mode module	Public, dynamic	The current multiplier of the score
baseScore	integer	Attribute of the Hard Mode module	Private, static	The base amount that the score increases by per second
playerMove	Boolean	Attribute of Hard Mode module	public, dynamic	Whether the player is currently able to move or not
jumpInterval	float	Attribute of	Public, static	The amount

		the Hard Mode module		that the player will jump per second (increase in y-coordinate per second of a jump)
spawnedPD	string	Attribute of the Hard Mode module	public, dynamic	The name of the power down that is about to be spawned
totalJumpLength	float	Attribute of Hard Mode module	Public, static	The amount of time that a jump will last
currentJumpLength	float	Attribute of Hard Mode module	private, static	The amount of time that has passed during the jump
coinsCollected	integer	Attribute of the Hard Mode module	Public, dynamic	The amount of coins collected within that 1 game
abilityActive	boolean	Attribute of the Hard Mode module	Private, dynamic	Whether or not an ability is active
spawnedAbility	string	Attribute of the Hard Mode Module	private, dynamic	The name of the ability being spawned into the game
currentAbility	string	Attribute of the Hard Mode module	Private, dynamic	Name of the ability active (if any)
playerAlive	boolean	Attribute of the Hard Mode module	Public, dynamic	Whether or not the player is alive
isFlying	boolean	Attribute of the Hard Mode module	Public, dynamic	Whether or not the player is flying
isPaused	boolean	Attribute of the Hard Mode module	Public, dynamic	Whether or not the game is paused
hmGameAcceleration	float	Attribute of the Hard Mode module	Private, static	The rate at which the speed of the

				game increases in hard mode
hmStartRate	float	Attribute of the Hard Mode module	Private, static	The beginning speed of the game
powerDownName	string	Attribute of the Hard Mode module	Private, dynamic	Name of the power down active (if any)
powerDownActive	boolean	Attribute of the Hard Mode module	Private, dynamic	Whether or not a power down is active
pdRemainingTime	float	Attribute of the Hard Mode module	Private, dynamic	Remaining time of the power down (if any)
abRemainingTime	float	Attribute of the Hard Mode module	Private, dynamic	Remaining time of the ability (if any)
obstacleCount	Integer	Attribute of Hard Mode Module	Public, dynamic	The number of obstacles active
tempCoinXPos	float (Vector2 in Unity)	Attribute of Hard Mode Module	Public, dynamic	The temporary storage for the x-value of the position of the randomly generated coins
tempCoinYPos	float (Vector2 in Unity)	Attribute of Hard Mode Module	Public, dynamic	The temporary storage for the y-value of the position of the randomly generated coins
obstMaxHeight	float	Attribute of Hard Mode Module	Public, dynamic	The maximum y-value that an obstacle can be generated at.
gameTimer	float	Attribute of Hard Mode Module	Public, dynamic	The amount of time the game has been running

2.6.10 - Game Summary Key Variables

Name	Data Type	Scope	Private/Public	Purpose
finalScore	float	Attribute of the Game Summary module	Public, dynamic	The final score after a player death
finalCoins	integer	Attribute of the Game Summary module	Public, dynamic	The final number of coins collected after a player death
highscore	float	Attribute of the Game Summary module	Public, dynamic	The current high score, retrieved from the external database
isNewHighscore	Boolean	Attribute of the Game Summary module	Public, dynamic	Whether or not the current score is a new highscore

2.7 - Key Data Structures

Name	Data Type	Scope	Private/Public	Purpose
arrSummary	1D Array	Attribute of the Game Summary Module	Public	To hold 3 values (coins collected, final score, game mode) for use in the game summary screen
dtbUsers	Access Database	N/A: outside bounds of program	Public	Stores all user data
arrPausePos	Float 1D Array	Attribute of Pause module	Public	Stores the current position of the user
arrPauseObst	Float 2D Array	Attribute of Pause module	Public	Stores the current position of the obstacles when paused

arrObstaclePos	Float 2D Array	Entire Program	Public	Stores current position of obstacles in-game
arrPowerUpNames	String 1D Array	Easy Mode & Hard Mode	Public	Stores the names of all the power ups
arrPowerDownNames	String 1D Array	Easy Mode & Hard Mode	Public	Stores the names of all the power ups

2.8 - Input Field Processes

2.8.1 - User Sign Up Details

Input Field	Process	Storage	Output
Email	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered within this field before submission <p>Format Check</p> <ul style="list-style-type: none"> - Data entered must follow the format of name@example.com 	User Details Table (Database)	<p>If Valid</p> <ul style="list-style-type: none"> - Display a tick on-screen next to the email field <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to enter a valid email - Clear entry in email field
Username	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered within this field before submission <p>Code Check</p> <ul style="list-style-type: none"> - Data entered must only contain alphanumeric characters, not any special characters such as !, ?, <, etc. 	User Details Table (Database)	<p>If Valid</p> <ul style="list-style-type: none"> - Display a tick on-screen next to the username field <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to enter a valid username - Clear entry in username field
Password	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered 	User Details Table (Database)	<p>If Valid</p> <ul style="list-style-type: none"> - Display a tick on-screen next

	<p>within this field before submission</p> <p>Length Check</p> <ul style="list-style-type: none"> - Password should be at least 6 characters, but less than 16 characters 		<p>to the password field</p> <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to enter a valid password - Clear entry in password field and confirm password field
Confirm Password	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered within this field before submission <p>Code Check</p> <ul style="list-style-type: none"> - Data entered must be identical to that of the 'password' field 	User Details Table (Database)	<p>If Valid</p> <ul style="list-style-type: none"> - Display a tick on-screen next to the email field <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to ensure password fields match - Clear entry in confirm password field
Sign Up Button click	Transfer data to external database	Array of records	<p>If all other fields are valid:</p> <ul style="list-style-type: none"> - Output message stating that sign up has been successful <p>If at least one of previous fields are invalid</p> <ul style="list-style-type: none"> - Output message asking user to try again

2.8.2 - User Login Details

Input	Process	Storage	Output
Username	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered 	Array of data which will be used to compare to data	<p>If Valid</p> <ul style="list-style-type: none"> - If both username &

	<p>within this field before submission</p> <p>Code Check</p> <ul style="list-style-type: none"> - Data entered must only contain alphanumeric characters, not any special characters such as !, ?, <, etc. <p>Database Comparison</p> <ul style="list-style-type: none"> - Data entered from this field, stored in the array, will be compared to that of the 'username' field in an external database 	held in database	<p>password entered are valid and the login button is pressed, display a message stating that login successful</p> <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to enter a valid username - Clear entry in username field
Password	<p>Data Validation</p> <p>Presence Check</p> <ul style="list-style-type: none"> - Data must be entered within this field before submission <p>Code Check</p> <ul style="list-style-type: none"> - Data entered must only contain alphanumeric characters, not any special characters such as !, ?, <, etc. <p>Database Comparison</p> <ul style="list-style-type: none"> - Data entered from this field, stored in the array, will be compared to that of the 'password' field in an external database 	Array of data which will be used to compare to data held in database	<p>If Valid</p> <ul style="list-style-type: none"> - If both username & password entered are valid and the login button is pressed, display a message stating that login successful <p>If Invalid</p> <ul style="list-style-type: none"> - Display message on screen asking user to enter a valid password - Clear entry in username field
Login button click	Take array of comparison data and compare its data to that of the data held in the external database for player login details	Array of records	<p>If all other fields are valid:</p> <ul style="list-style-type: none"> - Output message stating that login has been successful <p>If at least one of previous fields are invalid</p> <ul style="list-style-type: none"> - Output message

			asking user to try again
--	--	--	--------------------------

2.9 - Database

2.9.1 - The Need for a Database

For AstroSprint, I have decided that I need to implement a database to hold the following data:

- **Emails** (string): For sign up & to contact user about any issues with their account
- **Usernames** (string): For login & to uniquely identify users
- **Passwords** (string): For login user authentication, to make sure that game accounts are secure
- **Character Choices** (string): For character menu & gameplay, so that if a user logs out and logs in again, their character choice will be saved and they won't have to select it again each time
- **Coin Balance** (integer): For game summary & abilities menu, to allow players to upgrade abilities for a longer duration. Saved in the database so that players can retain the amount of coins that they have collected, even after logging out and logging back in.
- **Ability Levels** (integer): For ability menu, to allow users to have longer duration for abilities. Needs to be saved to ensure that users can retain the upgrades they have purchased, even after logging out and logging back in
- **Highscores** (integer): For showing in game summary, to let users have a sense of competition with themselves. Needs to be saved to ensure that users can always see their highscores and try and beat them, even after logging out and logging back in.

All of this table will be put in a database named 'dtbUsers', with each user having a unique username, & email. The username acts as a unique identifier of each user, and will be the primary key of the table. This is to make sure that each user can be uniquely identified and signed into, to prevent collisions and make sure that multiple users don't have access to the same account. This also prevents passwords from getting overwritten during sign up.

The variables in the table that don't need to be unique are: passwords, character choices, coin balance, ability levels & scores. Passwords don't have to be unique because passwords are not identifiers within themselves and, within login, they are only used to check to see if it correctly matches the password saved to the corresponding user in the database. Within sign up, having unique passwords also isn't necessary, as it is only saved to the corresponding user that is typed into the username input.

Having unique highscores, character choices, coin balances and ability levels is also not necessary, as these are things that all users need to select, earn or upgrade, and these overlapping between users wouldn't cause issues,

as it is not used as a form of identification within the program. It would also not be an issue because after login, these variables only have 1 value each (for the logged in user) and so it would not cause any collisions (or multiple pieces of data being accessed at once).

2.9.2 - Database Implementation

For creating a database within my game, I will be utilising Microsoft Playfab. This is a backend platform for building and managing online games, and it offers a range of APIs for developers to interact with its features

I will use Playfab's Realtime Database to store runtime data, such as highscores, and then make sure these are stored within the correct location (e.g. with the corresponding username) using the tools within the Playfab Software Development Kit.

- I will also utilise the unique Playfab commands within my code to be able to read and write other forms of data within my project.

2.10 - Usability Features

2.10.1 - Colour

Colour Palette

- I have chosen a dark colour palette for the background of AstroSprint menus, and very bright fonts for the forefront of AstroSprint. I have done this because I believe that it increases readability, and visibility on the screen.
- In-game, I have chosen to make the background a gradient of colours, depending on whether the player has chosen hard mode or easy mode

The colour palette for the background of easy mode is shown in the photo on the left, and the colour palette for the background of hard mode shown on the right.



When these are implemented into AstroSprint, I will use them as a gradient background. This is because I believe that it will be the best for visibility, and with the brightest colour at the bottom, it will be a good contrast to the foreground, which will be a black (colour hex #000000) platform that is 1/6th of the screen at the bottom.

The colour pallet for all of my in-game menus is as follows:

FFFFFF

000000

BAB9B9

EAE9E9

1C7E49

197CA8

I have chosen the black, whites & greys as a large contrast from each other to be able to easily distinguish between different buttons in-game versus the background. I have also selected the green & blue colours for the earth design at the bottom of the menu screens to make them look more vibrant and eye-catching.

- However, due to accessibility, I have not included these colours within important aspects that players need to interact with

Accessibility to Colour Blindness

- Due to needing to accommodate accessibility, I have chosen to not make the green & blue colours a part of any major aspect that players need to be able to understand and interact with, as they are colours that can be difficult to differentiate between for those suffering with colour blindness, especially those whose ability to see green & blue are impaired
- Another way I have helped to improve accessibility to those who are colour blind, is by keeping my colour scheme very contrasting, with very obvious bright vs dark colours on important aspects that players need to interact with, so that those with visual impairments will be able to distinguish between elements without relying on colour alone.

2.10.2 - Layout

Consistent Layout

- Within AstroSprint, I have decided to keep a consistent layout between all menu screens, so on a fundamental level, they look almost the same. This is for easy identification as to where the user is within the game loading process, as the gameplay screens look completely different to the menu screens

Buttons

- I have utilised rounded buttons within AstroSprint to be able for the player to be able to clearly navigate throughout the game and clearly be able to provide inputs into the program
 - For example, I have included 'log in' and 'sign up' buttons to be able to provide a clear interface for the player to be able to submit the data they have input. This lessens confusion about how to enter data, such as needing to guess what button to press top be able to submit data

Positioning

- I have positioned all menu text & buttons within the centre of the screen
 - I have done this because I believe that it makes the screen seem more cohesive and more aesthetically pleasing for the user, as well as bringing a balance within the menu and promoting symmetry, which brings the game together
 - I have placed all submission buttons below the text input areas, to indicate that all fields must be filled before submission
 - Validation techniques outlined above also make sure of this, however, the position helps to convey the message to the user before they are met with the error message

Text Input

- I have inserted the name of the field to be filled out within the text input box. I have done this to clearly signpost what data should be input within the fields
 - I have put this within the actual input box because I believe that putting it anywhere else would cause confusion as to which box it corresponds to. This method is the most effective at clearly showing what data goes within each input box

2.10.3 - Text

Font

- I have chosen 'Courier New' as my in-game font.
 - I have chosen this because I believe that it has the best readability due to it being monospaced. This means that every single letter has distinct features from one another, so there is no confusion as to what certain letters are, which is a common

issue in other fonts where a capital 'I' looks the same as a lowercase 'l'

- This is effective as it prevents confusion between letters and allows clear communication to the user within the game.

Colour

- I have chosen to use black text on light backgrounds and white text on dark backgrounds
 - I have done this because I believe that it provides the most contrast, which makes it the text clearer for the user

2.10.4 - User-Friendly Approach

Help Section

- I have included a tutorial screen on how to play the game, which includes the controls. This can be viewed around various points of the game, and so are convenient to access for users.

Feedback Mechanisms

- For each button, in order for the user to recognise what button they are about to click, the button changes colour slightly when the cursor is hovering over it.
 - This was discussed in my analysis, and it continues to be a feature I will include because of its user-friendliness.
- When a user selects a character to use in-game, it will also be shown to them in a separate location in order to make it clear what character has been selected, as well the button itself changing to a different state to show that it has been selected
- Written Game Information: The names of currently active power ups and their durations, the current score and the amount of coins collected will be shown on-screen to make it clear to the user what is going on in-game.

2.11 - Iterative Development Test Planning

2.11.1 - Sign Up Module Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Email Input Box	18	1	null	Erroneous When no data is entered for the email, it should not allow the user to create an account	User should be alerted to input an email
		2	someone.com	Erroneous This does not follow	User should be alerted to put in an

				the email format, and should therefore be rejected as an input	email with the correct format
		3	someone@example.com	Typical This is an email in the correct format, and therefore should be accepted	User should not be alerted.
		4	someone@example.co.uk	Boundary This is a valid email address, but is boundary as it ends with '.co.uk' as opposed to '.com'	User should not be alerted.
		5	testmail@gmail.com and then again with testmail@gmail.com	There should not be able to be 2 accounts linked to the same email	The user should be alerted that this email has been taken
Username Input Box	19	6	null	Erroneous When no data is entered for the main, it should not let the user create an account	User should be alerted to enter a username
		7	user??!!\$\$	Erroneous This data contains special characters, which should not be accepted by the program.	User should be alerted to only enter a username with alphanumeric characters
		8	testuser	Typical This is a valid username, and should be accepted by the program.	User should not be alerted.
		9	test.user	Boundary This data contains a '.', which should not be accepted, but is widely used within usernames for many sites.	User should be alerted to select a username without any special characters

		10	thename and then thename again	Although these are valid usernames, the user should not be able to choose a username that already exists in the database	User should be alerted that this username is not available
Password Input Box	20	11	null	Erroneous When there is no data entered for the password, it should not let the user create an account	User should be alerted to enter a password
		12	password	Boundary This password is 8 characters, which is the lower boundary for password length. This should be accepted by the program.	User should not be alerted
		13	password12345678	Boundary This password is 16 characters, which is the upper boundary for password length. This should be accepted by the program.	User should not be alerted
		14	typicalpass	Typical This password is 11 characters long, which should be accepted by the program.	User should not be alerted
		15	short	Erroneous This password is too short, as it is 5 characters long. This should not be accepted by the program.	User should be alerted to pick a password between 8 and 16 characters (inclusive)
		16	longerpassword1234	Erroneous This password is too	User should be alerted to pick a

				long, as it is 18 characters. This should not be accepted by the program.	password between 8 and 16 characters (inclusive)
Confirm Password Input Box	21	17	null	Erroneous When no data is entered, it should not let the user create an account	User should be alerted to input a confirm password
		18	typicalpass [identical to previously entered password]	Typical This password is identical to one that is previously entered in the 'password' field, so should be accepted	User should not be alerted
		19	notthesamepass	Erroneous This is not the same password as the one previously entered. This should not be accepted by the program.	User should be alerted to make the confirm password the same as

2.11.2 - Login Module Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Username Input Box	4	20	null	Erroneous When no data is entered for the username, it should not allow the user to log in	User should be alerted to input a username
		21	fakeuser	Erroneous This is not a username currently held in the database. This should not be accepted by the program.	userValid = False and User should be alerted of this.
		22	testuser	Typical	userValid =

				This is a username that is held in the database. This should be accepted by the program.	True and User should not be alerted (unless password is invalid).
Password Input Box	4	23	null	Erroneous When no data is entered for the password, it should not let the user log in	User should be alerted to enter a password
		24	invalidpass	Erroneous This data is not held in the external database. This should not be accepted by program.	passValid = False and User should be alerted of this.
		25	typicalpass	Typical This is a valid password, and should be accepted by the program.	User should not be alerted

2.11.3 - Character Menu Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Initial Character Selection Display	26	26	N/A	When the menu is opened for the first time by a user, the male astronaut should be selected and the other characters not.	Male astronaut appears next to the 'Your Icon' arrow
New Character Selection	24	27	Click on 'Select' for female astronaut	Typical When the 'select' button next to the female astronaut is clicked, the female astronaut character must be selected.	Female astronaut button changes from 'select' to 'selected'. Character next to 'your icon' changes to female astronaut

		28	Click on 'Select' for male astronaut	Typical When the 'select' button next to the male astronaut is clicked (after it has been changed to a different icon), the male astronaut character must be selected.	Male astronaut button changes from 'select' to 'selected'. Character next to 'your icon' changes to male astronaut
		29	Click on 'Select' for the star	Typical When the 'select' button next to the star is clicked, the star character must be selected.	Star button changes from 'select' to 'selected'. Character next to 'your icon' changes to star
		30	Click on 'Select' for the alien	Typical When the 'select' button next to the alien is clicked, the alien character must be selected.	Alien button changes from 'select' to 'selected'. Character next to 'your icon' changes to alien
		31	Click on random part of the screen with no button	Erroneous If no specific button is clicked, nothing should happen.	Menu remains in the same state that it was before the click happened.

2.11.4 - Tutorial Screen Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Back Button	12	32	Back button pressed	Pressing a button should lead to the execution of the 'GoBack()' function	User is taken back to play menu
		33	Random (non-button) area of screen pressed	The pressing of a random area of the screen is not linked to any function, and	Nothing happens

				therefore nothing should happen when this is pressed	
--	--	--	--	--	--

2.11.5 - Abilities Menu Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Initial Display of Power Up Level	25	34	N/A	When the menu is opened, it should show all of the power ups at level 1 with 30s displayed on the only light green box	1 light green box with '30s' on it, displayed to the player
Purchase of power up	25	35	Click purchase button on double jump ability with no coins in balance	Erroneous As the coin balance is 0, the price of the purchase is above the balance. This should mean that the purchase doesn't go through.	User should be alerted that they don't have enough coins for this purchase. Double jump level should not be increased, and display remains the same.
		36	Click purchase button on double jump ability with 80 coins in balance	Typical The coin balance is 80 coins, this is above the price of the first upgrade of the jump ability. This means that the purchase should go through.	Double jump level should increase by 1, have 2 light green boxes, show 40s on the rightmost green box, remove 50 coins from the balance and display 100.
Display of Coin Balance	22	37	N/A	When the abilities menu is initially opened, the player's current coin balance should be displayed on screen so they are	Player's coin balance displayed on screen

				able to see what purchases they can afford	
		38	Click on 50 coin purchase button with 100 coins in balance	When the player purchases an upgrade, the amount of coins in their coin balance should decrease accordingly, the updating of this balance shows the user what they can and can't afford.	Coin balance displays 50 coins

2.11.6 - Gameplay Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Foreground Movement	38	39	N/A	During the entire duration of gameplay (while the game is not paused), the foreground should be moving horizontally from right to left.	Movement of foreground from right to left.
Player Movement	38	40	'Space' key pressed	Typical When the flying ability is not active, pressing the space key should make the player jump.	Player jumps
		41	Non-binded key pressed: 'L'	Erroneous This is not a valid input, as there is no instruction that includes the use of the 'L' key to trigger an action. This should not have any effect on the game.	Nothing happens.
Obstacle Generation	48	42	N/A	When gameplay begins (as long as the game isn't paused), an obstacle appears every 5 seconds.	After 5 seconds have passed in-game, an obstacle appears

		43	N/A	When the flying ability is active, the obstacles should be able to spawn at increased y-values	Obstacles spawning at higher y-values than they would for regular movement
		44	Obstacle out of frame	When an obstacle is out of frame, it should be deleted.	Obstacles out of frame are deleted from the Hierarchy menu in Unity
Double Score Power Up	32	45	Player collision with double score power up icon	After collision, score should increase at double the rate it usually does. This should be a rate of 20 points per second, as opposed to the usual 10	Score should increase by 200 in 10 seconds.
		46	Waiting 30 seconds	As the power up is on level 1, after waiting 30 seconds, the score should go back to increasing at a rate of 10 points per second	After waiting 30 seconds for power up to wear off, the score should increase by 100 in 10 seconds.
Flying Power Up	32	47	Space key held down	With this power up active, the up arrow is binded to a control, the movement of the player horizontally upwards.	The player should move up as long as the button is being held
		48	Waiting 30 seconds	When on level 1, after 30 seconds have passed, the power up should become inactive	Player falls to the floor
		49	Player collision with power up icon	To activate the power up, the player needs to collide with the icon	Player is able to fly
		50	Space key held pressed when player is at top of screen	Even with flying active, the player should not be able to escape the frame of the game	Player is stopped from moving out of frame.

Double Coins Power Up	32	51	Player collision with coin when power up is active	After collision, the player's coin balance should increase by 2 after colliding with a coin, rather than 1	Player collects coin and gains 2 instead of 1
Double Jump Power Up	32	52	'Space' key pressed twice without touching the floor	When this power up is active, the player should be able to jump twice without touching the floor when a valid key is pressed (the 'space' key)	Player jumps twice.
Half Score Power Down	36	53	Player collision with half score power down icon	After collision, score should increase at half the rate it usually does. This should be a rate of 5 points per second, as opposed to the usual 10	Score should increase by 50 in 10 seconds.
		54	Waiting 45 seconds	As the power down duration is 45 seconds, after waiting 45 seconds, the score should go back to increasing at its usual rate	After waiting 45 seconds for power up to wear off, the score should increase by 100 in 10 seconds.
Blurry Screen Power Down	36	55	Player collision with blurry screen power down	When the player collides with a power down, it should be immediately activated.	Screen becomes blurry for 45 seconds
		56	Waiting 45 seconds	The power down duration is 45 seconds, so after waiting this, the screen should go back to normal	After 45 seconds, screen returns to normal state
Darkness Power Down	36	57	Player collision with darkness power down	When the player collides with a power down, it should be immediately activated.	Screen becomes dark for 45 seconds
		58	Waiting 45 seconds	The power down duration is 45 seconds, so after	After 45 seconds, screen

				waiting this, the screen should go back to normal	returns to normal state
Inverted Colours Power Down	36	59	Player collision with inverted colours power down	When the player collides with a power down, it should be immediately activated.	Screen colours become inverted for 45 seconds
		60	Waiting 45 seconds	The power down duration is 45 seconds, so after waiting this, the screen should go back to normal	After 45 seconds, screen colours returns to their normal state
Player death	36	61	Collision with obstacle	In AstroSprint, the player has 1 life. When the player hits an obstacle, they should die.	Player dies.
Pausing the Game	42	62	Pressing pause button	In AstroSprint, when the game is paused, the player should be shown the pause menu to be able to view the tutorial, exit the game or return to the current game	The player is shown the pause menu and all of its options
		63	Pressing unpause button	When the unpause button is pressed, the player should be able to go back to the gameplay	Player returns to the gameplay
		64	Pressing area of screen not linked to a button	Pressing an area on the pause menu should not have a linked function, and so nothing should happen	Nothing
		65	Pressing the tutorial button	The function of the tutorial button is so that the player can press it and see a tutorial for the game while the game is paused. This should be shown when the tutorial button is pressed	Tutorial image is shown on screen

		66	Pressing the pause button, then pressing back	In-game statistics, current power ups, and current locations of obstacles should be maintained while the game has been paused, so when the game is unpause, the game returns to the state it was in before the pause.	Game statistics are maintained
--	--	----	---	---	--------------------------------

2.11.7 - Game Summary Testing

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Exit To Play Menu Button	12	67	Pressing the exit to play menu button	Pressing the button labelled 'exit to play menu' should take the player back to the play menu	Player is taken to the Play Menu
		68	Pressing anywhere on the screen that is not a button	Pressing an empty space on the screen should have no effect, because there is no code attached to it	Nothing
Display of Final Score	50	69	Finishing a game	After a game has been played, it is essential that the final score is shown to the player so they know how well they have done	Final score displayed on screen
Display of Final Coins	50	70	Finishing a game	After a game has been played, it is essential that the final coins is shown to the player so they know how many coins they have gained this game	Final Coins displayed on screen

2.12 - Post Development Test Planning

Feature to Test	Success Criteria No.	Test No.	Method of Testing	Justification of Test Method	Expected Outcome
All menus titled	1	1	Go through of all menus with the title shown	This ensures that every single title of the menus are shown, and menus where this is not able to be shown.	Every single menu is titled
Clearly labelled menus, with options within them being clear	2	2	Go through the full game, showing all of the menu titles and the options within them, like buttons leading to clear options	This shows all of the visual aspects of the menus and makes it clear which ones are/aren't fully clear	Every menu is clearly labelled with the options within them being clear
Main Menu UI	3	3	Video of main menu and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and therefore successful	The main menu has an interactable user interface, which is descriptive and fit for purpose
Login Menu UI	4	4	Video of login menu and all interactable features	By using all of the buttons and input fields on the menu, it shows that the user interface is fully interactable and works as intended	The login menu has an interactable user interface, which is descriptive and fit for purpose
Sign Up Menu UI	5	5	Video of sign up menu and all interactable features	By using all of the buttons and input fields on the menu, it shows that the user interface is fully interactable, and therefore successful	The sign up menu has an interactable user interface, which is descriptive and fit for purpose
Play Menu UI	6	6	Video of play menu and all interactable	By using all of the buttons on the menu, it shows	The play menu has an interactable

			features	that the user interface is fully interactable, and therefore works as intended	user interface, which is descriptive and fit for purpose
Tutorial Screen UI	7	7	Video of tutorial screen and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and therefore successful	The tutorial screen menu has an interactable user interface, which is descriptive and fit for purpose
Character Menu UI	8	8	Video of character menu and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and understandable	The character menu has an interactable user interface, which is descriptive and fit for purpose
Abilities Menu UI	9	9	Video of abilities menu and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and therefore successful	The abilities menu has an interactable user interface, which is descriptive and fit for purpose
Pause Menu UI	10	10	Video of pause menu and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and therefore successful	The pause menu has an interactable user interface, which is descriptive and fit for purpose
Game Summary UI	11	11	Video of game summary menu and all interactable features	By using all of the buttons on the menu, it shows that the user interface is fully interactable, and therefore successful	The game summary menu has an interactable user interface, which is descriptive

					and fit for purpose
Back buttons on all menus	12	12	Video showing use of back buttons on all menus	Clicking on all of the back buttons within the game to see if the user gets taken back to the previously selected menu is the only method to see if the back buttons work.	There is a back button present on every single menu, and it will take user back to the previous menu
Non-bright, cohesive colour scheme	13	13	Primary user shown video of gameplay and review	Ensures that the test is fair if it is done by an outside source (not me, the developer, who created the colour scheme) and the cohesiveness and brightness can be judged by inspection	The entire colour scheme of the game is cohesive
Varied environments, with users being able to play in places with different backgrounds	14	14	Video of user playing in all of the different environments	Showing all of the different environments being used in the game proves the existence of them	There are varied environments in the game that the player can play in
Login system	15	15	Video of login menu being used, and a pre-existing user being able to log in.	Ensures that the login system functions correctly and allows pre-existing users access to the rest of the game	There is a login system that allows pre-existing users to log in, and has input validation for all fields
Sign up system	16	16	Video of sign up menu being used and new user being able to sign up	Ensures that new users can sign up with valid credentials and access the game	There is a login system that allows new users to sign up, and has input validation for all fields

Catalogue of characters that the player can choose from	17	17	Video of character menu being used, showing all options for characters within them	Ensures that all planned options for characters are visible and able to be selected by the user	All character icons are visible and able to be selected
Input validation for email when signing up	18	18	Video of invalid and valid inputs being input into the email field and corresponding error messages	Ensures that input validation properly handles the invalid and valid inputs with the correct corresponding messages where necessary	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up
Input validation for username when signing up	19	19	Video of invalid and valid inputs being input into the username field and corresponding error messages	Ensures that input validation properly handles the invalid and valid inputs with the correct corresponding messages where necessary	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up
Input validation for password when signing up	20	20	Video of invalid and valid inputs being input into the password field and corresponding error messages	Ensures that input validation properly handles the invalid and valid inputs with the correct corresponding messages where necessary	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up
Comparison between password and confirm password fields	21	21	Video of passwords that match and don't match, and corresponding error messages for passwords that don't match	Ensures that the comparison between the two fields is present and prevents them from logging in if they don't match	Password fields that match are allowed to sign up. Password fields that don't match cause an appropriate error message to appear.
Coins balance retrieved	22	22	Video of coin balance being	Ensures that the coin balance is	Non-zero coin balance is

from external database in abilities menu			shown on-screen from a pre-existing user that just logged in	successfully retrieved from the external database	shown on-screen after a pre-existing user logs in
Coin balance updated in external database after every upgrade purchase	23	23	Video of user upgrading power ups using coins and screenshot of the coin balance in external database after these purchases have been made	Ensures that the power up upgrades subtract coins from both the in-game variable and the coin data held in the external database	Coin balance decreases after a purchase is made
Indication of which character was chosen in Character Menu	24	24	Video of a character being selected and its corresponding indicator showing an image of that character.	Ensures that there is a clear indication of what character has been selected, after a character has been selected	The icon of the character selected is shown separately on the screen
Indicator of each power up's level	25	25	Video of user opening the abilities menu, and being able to see, on inspection, see something that shows the power up's level	Ensures that the power up level indications are an accurate representation of how upgraded the power up is	All of the power ups have a clear indication of how much they have been upgraded
There is a default selected character	26	26	Video of a new user opening the character menu and there being an indicator showing their currently selected character, before they have selected anything themselves.	Ensures that there is always a character selected for the user even if they don't select one themselves	The male astronaut character is selected even before any buttons are clicked.
Implementing challenges to gain in-game currency	27	27	Video of challenges menu showing all of the different challenges, how much currency they are worth on completion, and the current	Ensures that the challenges are present, and it is clear to the user how to achieve them, and what they achieve by completing them	The challenges menu accurately and understandable displays all of the information about the in-game

			progress towards completing the challenge		challenges
Game being 2D	28	28	Video of every single menu and all of the gameplay and inspection on what planes the game exists on	Ensures that the entire game is 2D	The entire game is 2D
Slow escalation in difficulty in the easier mode of the game	29	29	Video of easy mode gameplay, showing the horizontal movement getting progressively faster	Ensures that there is an escalation of difficulty within the game, but it is slow	There is a slow escalation of difficulty within the game
Indication of what power ups are active to user	30	30	Video of gameplay, where there is a collision with a power up icon, with text shown at the top of the screen, naming the power up.	Ensures that the collisions between the player and the power up icons work appropriately	Collision with a power up icon causes the name of the power up to be shown at the top of the screen
Indication of power up remaining duration when active	31	31	Video of gameplay, where there is a collision with a power up icon, with a visual indicator showing the duration of the power up.	Ensures that the indicator of the duration is clearly shown on-screen to players, with a clear purpose	Duration of power ups indicated on-screen
Power-ups	32	32	Video of gameplay, showing the use of power ups	Ensures that power up effects are able to be used in-game	Power up effects are able to be used in-game
Power up duration according to level upgraded	33	33	Video of upgrading the power ups on the abilities menu, and then going to the gameplay and timing how long it takes for a power ups effects to stop working	Confirms that the power up durations are based on the level they are at	Power ups upgraded to level 1 last for 30 seconds, level 2 for 40 seconds, level 3 for 50 seconds, level 4 for 60 seconds and

					level 5 for 120 seconds.
Scoring system based on how long you last in-game	34	34	Video of gameplay, where scores are shown at the top right hand corner of the screen, and by inspection, seeing if the score increases as time passes, in a regular pattern.	Confirms that the scores is increased based on the time has passed	Score increases by 10 per second.
Different game modes	35	35	Video of gameplay where the user clicks on 2 different buttons to lead to 2 different game modes, that are clearly distinguishable	Ensures that the two game modes are completely separate and exist in 2 separate scenes	2 separate buttons lead to 2 separate game modes
Implementation of negative power ups in a harder mode	36	36	Video of gameplay, where a player collides with a negative power up, and gets the corresponding negative effect for 30 seconds.	Confirms that the power downs don't last for less or longer than intended, and their effects are actually applied after a collision	Negative effects of power downs are triggered when the player collides with the icon, and last for exactly 30 seconds
Implementation of a sharp escalation in difficulty in the harder mode, in comparison to the easy mode	37	37	Video of gameplay of easy mode next to a video of gameplay of hard mode, showing how the speed increase in hard mode is quicker than that of the easy mode.	Confirms that the escalation in difficulty is visibly different in easy mode and hard mode	Escalation of difficulty in easy mode is slower than that of hard mode
Game is endless	38	38	Video of gameplay, and on inspection, the foreground should be looping continuously through the same	Confirms that the foreground will continuously repeat without any breaks	Foreground loops without any breaks

			few options		
Random generation of in-game currency within the playing environment	39	39	Video of gameplay, and inspecting the generation of coins to see if they are generated at both random time intervals and at random positions on-screen	Ensures that the spawning of the coins is correctly randomised	The spawning of coins is random
Collection of in-game currency after colliding with it	40	40	Video of gameplay, where the player collides with a coin in-game and collects them, as shown in a text element on-screen	Ensures that the player is able to gain coins by colliding with them, and that collisions have the appropriate effect	Names of active power ups shown on-screen
Ability to upgrade power-ups outside of the playing environment	41	41	Video of abilities menu and of user using coins to upgrade the power ups	Ensures that power up upgrades are able to be purchased in-fame	The duration of the power up increases
Ability to pause in-game	42	42	Video of gameplay, where the user clicks the pause button and is then taken to the pause menu	Ensures that the player is able to pause all of the functions of the game at will	The entire game stops moving and the player is taken to the pause menu
Maintain current score after coming out of pause menu	43	43	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same score as they had before they paused	Ensures that pausing the game does not alter the score of the player in any way	The player is on the same score directly after unpausing
Maintains player's state (alive or dead) after pause	44	44	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same state (alive) as	Ensures that pausing the game does not alter the state of the player in any way	The player is still alive directly after unpausing

			they had before they paused		
Maintains current coins collected after pause	45	45	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same coins as they had before they paused	Ensures that pausing the game does not alter the number of coins collected in any way	Same amount of coins collected for the player after the game has been unpause
Maintains power up remaining duration after pause	46	46	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same power up duration remaining as they had before they paused	Ensures that pausing the game does not alter the remaining duration of the power ups in any way	Same duration left on the power ups after the game has been unpause
Player death when player leaves the view	47	47	Video of gameplay where the player leaves the view of the game and them being taken to the game summary menu after	Confirms that leaving the bounds of the camera causes the player to die	The player dies and is taken to the game summary menu.
Player death after collision with obstacle	48	48	Video of gameplay where the player collides with an obstacle and then is taken to the game summary menu after	Ensures that the obstacles and players are correctly interacting with each other	Player dies after colliding with an obstacle
Random generation of power ups	49	49	Video of gameplay, and inspecting the generation of individual power ups to see if they are generated at both random time intervals and at random positions on-screen	Ensures that the generation of power ups is actually random, by inspection	The generation of power ups in-game is random

After player death, in-game statistics shown in the game summary	50	50	Video of gameplay, where the player dies and is taken to the game summary menu, where accurate statistics from the previous game are shown in on that menu.	Ensures that statistics are successfully transferred to the game summary menu, with the data 11 being correct from the most recent game	All in-game statistics are successfully and accurately transferred from the gameplay to the external database
After player death, in-game statistics transferred to external database	51	51	Video of the abilities menu, where the player views the coin balance, then goes into the gameplay, collects coins, where the player dies and is taken to the game summary menu, exits and returns to the Play Menu, then new value of coin balance is checked in the external database to be increased from the original value.	Ensures that after a game where the player dies, all of their statistics are saved (and therefore the change due to this shown in the game)	Coin balance would have increased
Stores highscores	52	52	Screenshot of external database for a user showing the field that contains their highscore	Ensures that all players have a field that stores their high score	User data has a field that contains the high score
Stores coin balance	53	53	Screenshot of external database for a user showing the field that contains coin balance	Ensures that all players have a field that stores their coin balance	User data has a field that contains the coin balance
Stores selected character	54	54	Screenshot of external database for a user showing the field that contains their selected character	Ensures that all players have a field that stores their selected character	User data has a field that contains the selected character

Stores usernames from sign up	55	55	Screenshot of external database for a user showing the field that contains their username	Ensures that all players have a field that stores their username	User data has a field that contains the username
Stores passwords from sign up	56	56	Screenshot of external database for a user showing the field that contains their password	Ensures that all players have a field that stores their password	User data has a field that contains the password
Stores emails from sign up	57	57	Screenshot of external database for a user showing the field that contains their email	Ensures that all players have a field that stores their email	User data has a field that contains the email
Stores data from 1 user all in the same place	58	58	Screenshot of the external database showing all of the corresponding fields for all of the user data in one place	Ensures that all players have their data stored separately, with all of the corresponding fields for 1 user kept separately from the same fields for other players	User data split by username, with each user having their own database, separate from other users
Storage of power up levels for each player	59	59	Screenshot of external database for a user showing the field that contains the power up levels	Ensures that all players have a field that stores levels of power ups	User data has a field that contains the power up levels
Default power up level 1 (Duration 30s)	60	60	Screenshot of external database for new user showing that the default power up levels are all 1	Confirms that new users start with a power up of 1 for all power ups	New user's power up levels all have the value of 1
Default coin balance of 0	61	61	Screenshot of external database for new user showing that the default coin balance is 0	Confirms that new users start with a coin balance of zero	New user's coin balance has the value of 0
Default highscore of	62	62	Screenshot of external database	Confirms that new users start with a	New user's high score has

0			for new user showing that the default highscore is 0	high score of zero	the value of 0
---	--	--	---	--------------------	----------------

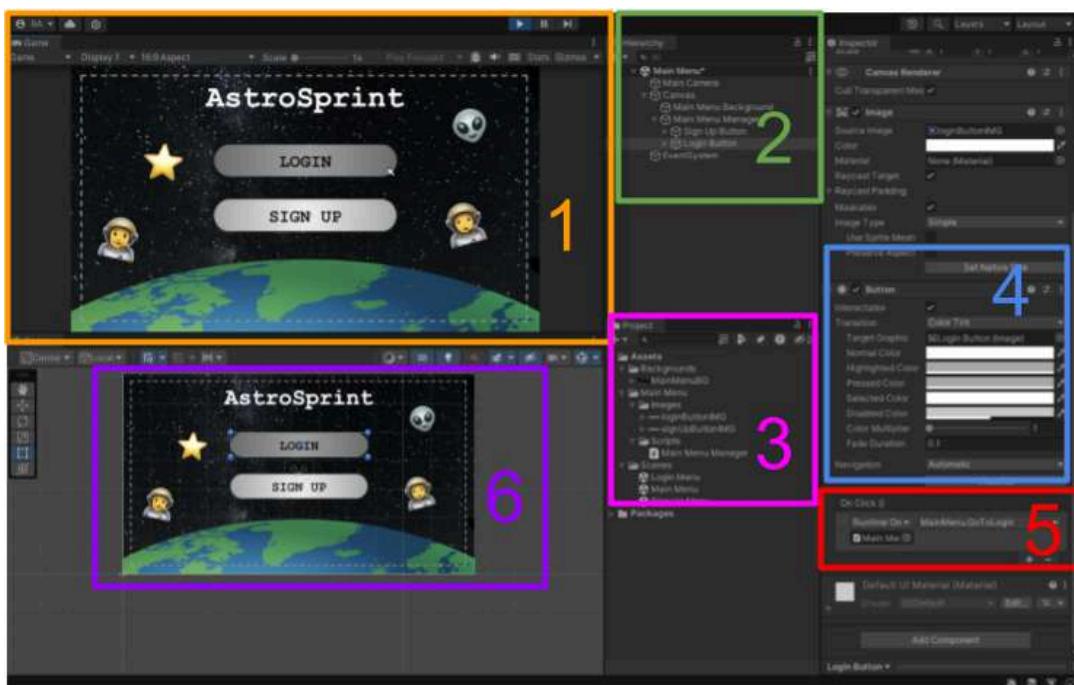
3 - First Iteration Development (AstroSprint v1.0)

3.1 - AstroSprint v1.0 Summary

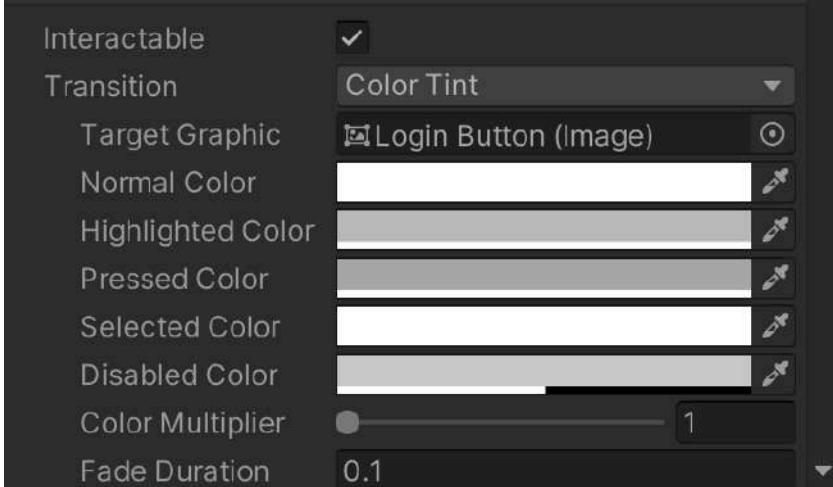
Below is the development of the first iteration of my code, which includes the Main Menu, Login Menu & Sign Up Menu. All together, these make up AstroSprint v1.0.

3.2 - Main Menu Development

3.2.1 - Main Menu Development Interface

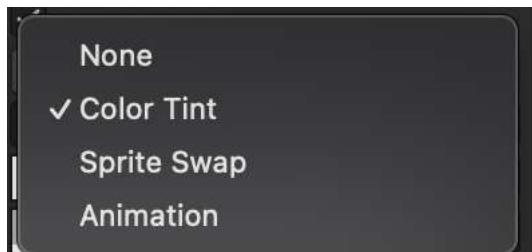


Section No.	Explanation
1	<p>This is the area where what is shown to the user during runtime is shown, known as the 'Game' view. When the game is first loaded, the main menu is the first thing shown, and this is what is shown to the user.</p> <p>I have included the user feedback mechanism of highlighting when a button is hovered on. This is shown by the mouse hovering over the login button, and it being darker than that of the button underneath it, which is not being hovered over. This was discussed in the analysis of 'Geometry Dash Lite' (1.3.1 - Existing Solutions) and was a feature I decided to implement into my game. I have done it slightly differently, making it so that the button becomes darker, rather than it being larger, like it does in 'Geometry Dash Lite'. I have</p>

	done this because it is much clearer visually than the enlargement of the button, that it is being hovered over.
2	<p>This is the Unity Hierarchy. This shows all of the game objects currently within the scene, both visible (the buttons & the background) and non-visible (The Main Menu Manager, camera, EventSystem, etc.) are contained in the hierarchy.</p> <p>The Main Menu Manager is a game object used for controlling the buttons on the main menu. It has a script (code) attached to it, which holds the subroutines for the login button and the sign up button</p>
3	<p>This is the project files manager in Unity. It shows all of the files currently active in my project. These files correspond to files on my device, and are held within a larger folder for the first prototype of my project: 'AstroSprint v1.0'</p> 
4	<p>This is a section of the 'Inspector' in Unity. The inspector "displays detailed information about your currently selected GameObject, including all attached Components and their properties" [Source: Unity Manual]</p>  <p>This part of the section shows the colour of the button when the mouse is hovering over it, the 'highlighted colour'. The highlighted colour is hex code BAB9B9 (as outlined in 2.11.1 - Colour). The pressed colour is slightly darker than this, of hex code A8A8A8. I have chosen this colour because it is slightly darker than BAB9B9, so it shows the difference</p>

between a button being hovered over and a selected button, without being too dark that it blends into the background or the text is illegible.

This is all able to happen due to the interactable box being ticked and the transition being 'Color Tint', which describes the method being used to show the interactions with the button.



Out of these options for the transition, I chose 'Color Tint' as it would be the most accessible option. This is for people with sight issues such as colour blindness, as they can perceive the change in brightness of colours. This was a limitation of my project, outlined in 1.1 - *Introduction*, and by choosing this option, I am reducing the amount of limitations within my project, and making it accessible to more users.

5

This section shows the 'On Click' function of the login button.

This is the name of the function that is called when the button is clicked on

This is a drop down option that describes when the login button will be able to be used. I have set this to be 'Runtime Only'



Settings for both the login button (displayed) and the sign up button are the same.

The 'Main Menu Manager' Script, which belongs to the Main Menu Manager game object and is attached to the login & sign up buttons to provide them with function by allowing them to access the subroutines within the script

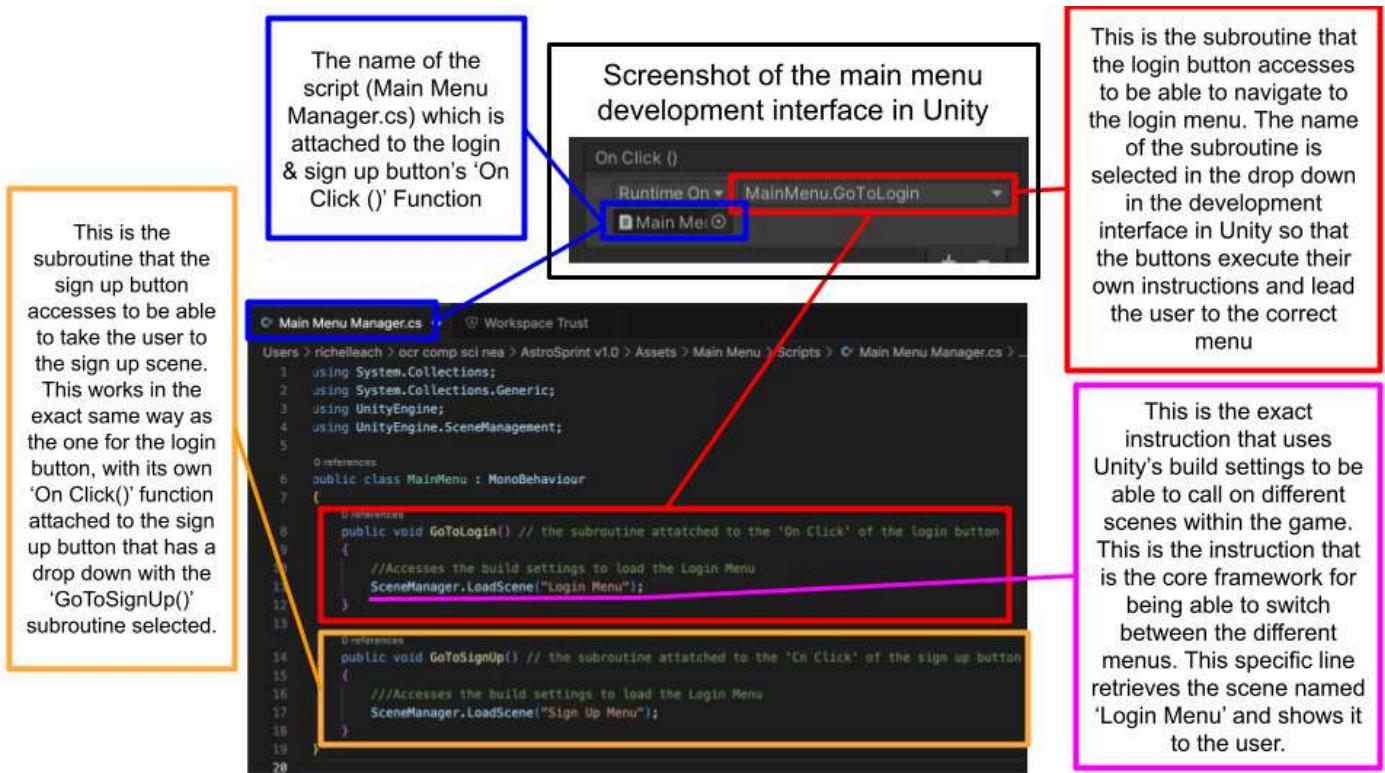
This is the name of the function that this button will utilise. This has been selected as the subroutine within the 'Main Menu Manager' script, which allows the user to be taken to the login menu when they click on the login button

I have chosen for the drop down option to be 'Runtime Only'. I have done this to retain the functionality of the button (the function is not called when 'off' is selected), while



	reducing the load on the system by making it not have to support the function when the game is not being run (as I can use the hierarchy and files to navigate the different scenes in the game, so do not rely on the buttons outside of testing)
6	This aspect of the development interface for the main menu is the scene view. This allows me to see everything inside and outside of what is being shown to the user in the game.

3.2.2 - Main Menu Code



I have made each button its own subroutine to reduce the complexity of the code and make it easier to debug. If I had done this in an alternative method, such as using if statements to see which button had been pressed, then the code would be much longer, and would be harder to test individual functions. This would make the program less maintainable, as it would be harder to update individual buttons without affecting the other ones. It would also make the code harder to debug, as you cannot test each button individually to see where issues arise. The use of subroutines within my project is essential for creating a maintainable game with understandable code.

To add to the maintainability of my code, I have added comments above less self-explanatory aspects of the code. This ensures that anyone who reads my code understands what each aspect of it does. This is also useful for when I debug my code after testing it, as I can see what the specific functionality

is of each section of code and be easily able to see what section causes certain issues.

Screenshot of the build settings in Unity

Build Settings

Build Settings

Scenes In Build

- ✓ Scenes/Main Menu
- ✓ Scenes/Login Menu
- ✓ Scenes/Sign Up Menu

Build Settings

Build Settings

Scenes In Build

- ✓ Scenes/Main Menu
- ✓ Scenes/Login Menu
- ✓ Scenes/Sign Up Menu

0
1
2

This is the aspect of the build settings that corresponds to the login menu. The name held in here corresponds to the name called within the login subroutine. The same applies to the 'Sign Up' menu, where the corresponding menu name from the build settings is used to call upon this scene for it to be shown to the user

These are the respective indexes of all of the different scenes within AstroSprint. They are automatically assigned to any scene added to the build settings

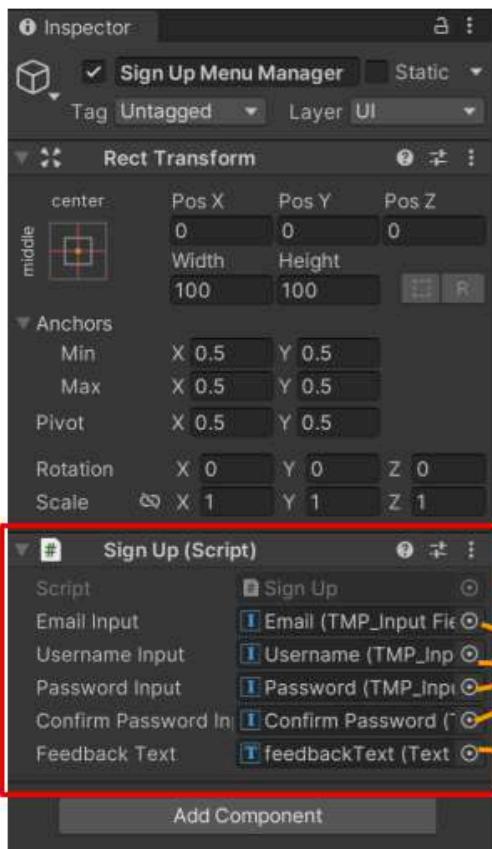
I have chosen to call my scenes to be shown to the user by their name, rather than their index. I have done this mainly so that the code is clear, and is understandable to others without having to go into the build settings to see the corresponding menu name, making the code overall more maintainable. I have also done this to ensure that the correct menu is always called, even if by chance the order of them in the build settings changes (therefore changing the index numbers).

The un-annotated code is shown below:

```
C# Main Menu Manager.cs X  Workspace Trust
Users > richelleach > ocr comp sci nea > AstroSprint v1.0 > Assets > Main Menu > Scripts > C# Main Menu Manager.cs > ...
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class MainMenu : MonoBehaviour
7  {
8      public void GoToLogin() // the subroutine attached to the 'On Click' of the login button
9      {
10          //Accesses the build settings to load the Login Menu
11          SceneManager.LoadScene("Login Menu");
12      }
13
14      public void GoToSignUp() // the subroutine attached to the 'On Click' of the sign up button
15      {
16          //Accesses the build settings to load the Login Menu
17          SceneManager.LoadScene("Sign Up Menu");
18      }
19 }
```

3.3 - Sign Up Menu Development

3.3.1 - Sign Up Menu Game Objects



This is the inspector for the 'Sign Up Menu Manager'. This empty game object controls the entire 'Sign Up' menu screen and all of its functions.

To achieve this, a script (set of code) called 'Sign Up' is attached to this game object. This script contains all of the different subroutines for the different input boxes, their respective validation routines, and what to do if the sign up is or isn't successful.

It's because of this script, that the data entered can be validated and the user alerted of any issues, or send the data to an external database (Microsoft PlayFab)

These boxes are where the inputs mentioned in the script are attached to the actual input box game objects. This has to be done in order for the script to access the data that is entered into the boxes, and carry out the necessary checks on each.

The feedback text is also attached to this, so that the appropriate messages can be shown to the user.

3.3.2 - Sign Up Menu Code

This is the beginning section of the code for the Sign Up Menu, lines 1 to 19.

```

1 // importing libraries to use in the code
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6 using PlayFab;
7 using PlayFab.ClientModels;
8 using System.Text.RegularExpressions;
9 using TMPro;
10
11 //the class that allows the user to sign up
12 public class SignUp : MonoBehaviour
13 {
14     //creates input fields for all the different sign up details
15     public TMP_InputField emailInput;
16     public TMP_InputField usernameInput;
17     public TMP_InputField passwordInput;
18     public TMP_InputField confirmPasswordInput;
19     public TMP_Text feedbackText; // A UI text element for feedback messages

```

1

2

Section No.	Explanation
1	<p>This is Lines 1-10 of the sign-up menu code. This imports various libraries from Unity, to be used within my code. These libraries have their respective uses:</p> <ul style="list-style-type: none"> - System.Collections & System.Collections.Generic: Allow the use of complex data structures, such as arrays & lists - UnityEngine: Imports the use of Unity's core engine functionalities, such as accessing GameObjects - UnityEngine.UI: Imports the use of Unity's UI functionalities, allowing access to user interface objects (e.g. the feedback messages on the screen) - PlayFab: Imports the PlayFab namespace, which is part of PlayFab SDK used for managing backend services like the user database - PlayFab.ClientModels: Provides specific classes & structures used to define & handle data models for PlayFab operations on the client side - System.Text.RegularExpressions: Allows the use of Unity's regular expression functions for validation (e.g. format checks) <ul style="list-style-type: none"> - This is essential for my format checks for the validation of the username (only alphanumeric characters) - TMPro: Text Mesh Pro allows the script to handle advanced text rendering and manipulation for better quality UI text elements
2	<p>This is lines 11-19 of the sign up menu code. This initialises all of the input fields, allowing them to be accessed and</p>

validated within the code.

Feedback text is also initialised here, which is a UI text element that sends messages to the user in event that there's an issue with their inputs, or if their registration is successful.

Lines 21 to 55
of Sign Up
Code

```
21 public void OnSignUpButtonClicked()
22 {
23     // allowing the user to input all of the different sign up fields
24     string newEmail = emailInput.text;
25     string newUsername = usernameInput.text;
26     string newPassword = passwordInput.text;
27     string pass2Confirm = confirmPasswordInput.text;
28
29     // checks if there is an input in the email field
30     if (string.IsNullOrEmpty(newEmail))
31     {
32         feedbackText.text = "Please enter an email";
33         return;
34     }
35
36     // checks if there is an input in the username field
37     if (string.IsNullOrEmpty(newUsername))
38     {
39         feedbackText.text = "Please enter a username";
40         return;
41     }
42
43     // checks if there is an input in the password field
44     if (string.IsNullOrEmpty(newPassword))
45     {
46         feedbackText.text = "Please enter a password";
47         return;
48     }
49
50     // checks if the password is between 8 and 16 characters
51     if (newPassword.Length < 8 || newPassword.Length > 16)
52     {
53         feedbackText.text = "Ensure password is between 8 and 16 characters";
54         return;
55     }
}
1
2
3
4
5
```

Section No.	Explanation
1	The first section of code (lines 21 to 28) include the declaring of the variables newEmail, newUsername, newPassword & pass2Confirm. These variables are declared to be the input in their respective textbox in-game. This corresponds to success criteria number 16 ('Sign Up system'), as this declares all of the input variables, allowing the user to begin the sign up process. This also links to the success criteria points 18, 19, 20 & 21, which are the points linking to the validation of sign up menu inputs. The declaration of these variables allows the user's inputs to be used later on in the validation process.

2	The next section (lines 29 to 34) of this code is the presence check for the email field. This is just checking if there is anything inputted in the email field, before attempting to validate. It utilises a function (IsNullOrEmpty) that checks if the field is empty, and uses the UI text element 'feedbackText' to tell the user to input an email if needed.
3	The next section (lines 35 to 41) of this code is the presence check for the email field. This is just checking if there is anything inputted in the username field, before attempting to validate. It utilises a function (IsNullOrEmpty) that checks if the field is empty, and uses the UI text element 'feedbackText' to tell the user to input a username if needed.
4	The next section (lines 42 to 49) of this code is the presence check for the password field. This is just checking if there is anything inputted in the password field, before attempting to validate. It utilises a function (IsNullOrEmpty) that checks if the field is empty, and uses the UI text element 'feedbackText' to tell the user to input an password if needed.
5	The last section (lines 50 to 55) of this code is a length check for the password field. It ensures that the password is both less than 16 characters and longer than 8 characters (inclusive). If it isn't, the UI text element is used to alert the user to enter a password between 8 and 16 characters (inclusive)

Lines 57 to 88
of Sign Up
Code

```

57     // checks if the value entered in the confirm password field is the same as the password field
58     if (newPassword != pass2Confirm)
59     {
60         feedbackText.text = "Please ensure that passwords match.";
61         return;
62     }
63
64     // uses the email format subroutine to ensure email is in the correct format
65     if (!IsValidEmail(newEmail))
66     {
67         ShowFeedback("Invalid email format. Please enter an email in the format 'someone@example.com'");
68         emailInput.text = ""; // clears the email field
69         return;
70     }
71
72     // uses the alphanumeric subroutine to ensure username only contains alpha numeric characters
73     if (!IsAlphanumeric(newUsername))
74     {
75         ShowFeedback("Username can only contain alphanumeric characters."); // alerts the user of their error
76         usernameInput.text = ""; // clears the username field
77         return;
78     }
79
80
81     // Create a PlayFab registration request
82     var registerRequest = new RegisterPlayFabUserRequest
83     {
84         Email = newEmail,
85         Username = newUsername,
86         Password = newPassword,
87         RequireBothUsernameAndEmail = true
88     };

```

1

2

3

4

Section No.	Explanation
1	The first part of the code once again validates the password, through making the user re-type their password. This code first checks if the password entered in the 'confirm password' field is the same as the one entered in the 'password', and if they don't, then the user is alerted through the feedback text game object. If they do match, then the code continues on. This links to success criteria point 21, 'Comparison between password and confirm password fields', as this subroutine does exactly what it describes. This also links to success criteria 20 ('Input validation for password when signing up'), as this is an essential & final aspect of validation of the password before it goes back to the external database. This subroutine also makes up a part of fulfilling criteria number 16 ('Sign up system') as this is a part of the process before the user can fully sign up to the system.
2	This next section of code is the subroutine used to validate the email input. This calls a subroutine called 'IsValidEmail' (declared later on) and if the subroutine processes the email as being in the incorrect format, the user is alerted of this. This links to success criteria number 19, the validation of email, as it handles most of this by calling the validation subroutine and informing the user if what they have inputted is invalid.

3	This next section of code is the subroutine used to validate the username input. This calls a subroutine called 'IsAlphanumeric' (declared later on) and if the subroutine processes the username as having special characters (as only alphanumeric ones are allowed), the user is alerted of this. This links to success criteria number 19, the validation of username, as it handles most of this by calling the validation subroutine and informing the user if what they have inputted is invalid.
4	The last section of this code handles creating a new record in the external database (PlayFab) for the new player. This happens after all of the validation, to ensure that only correct valid data is sent to the external database. This links to success criteria numbers 55, 56 & 57, which include storing the email, username and password in an external database. This fulfills these criteria because, by making a new registration request, it means that Unity is sending my player input data to Playab (the external database for my game).

Lines 90 to 126
of Sign Up Code

```
90     // Send the registration request to PlayFab
91     PlayFabClientAPI.RegisterPlayFabUser(registerRequest, OnRegisterSuccess, OnRegisterFailure);
92 }
93
94     private bool IsValidEmail(string email)
95     {
96         // Using the regex pattern for emails to create a subroutine for basic email validation
97         string emailPattern = @"^[\w\.-]+@[^\w\.-]+\.\w{2,}$";
98         return Regex.IsMatch(email, emailPattern);
99     }
100
101     private bool IsAlphanumeric(string username)
102     {
103         // Using the regex pattern for emails to create a subroutine for basic email validation
104         string alphanumericPattern = @"^[\w]{3,16}$";
105         return Regex.IsMatch(username, alphanumericPattern);
106     }
107
108     public void OnRegisterSuccess(RegisterPlayFabUserResult result)
109     {
110         // tells the user that their registration was successful
111         feedbackText.text = "Registration successful!";
112     }
113
114
115     public void OnRegisterFailure(PlayFabError error)
116     {
117         // tells the user that there's been an error
118         feedbackText.text = "Error: " + error.ErrorMessage;
119     }
120
121     public void ShowFeedback(string message)
122     {
123         feedbackText.text = message;
124         feedbackText.gameObject.SetActive(true);
125     }
126 }
```

This section of code first sends the registration request (that was formed in the previous section) to PlayFab (Section 1). It then declares the validation routines for the email (Section 2) & username (Section 3) fields. The validation of the email links to success criteria number 18, the validation of email, as it handles the comparison to Unity's RegEx pattern for emails to be in the format of 'someone@example.com'. This subroutine is called earlier in the code in order for an email entered to be validated. The validation of the username links to success criteria number 19, 'Input validation for username when signing up'. 'IsAlphanumeric' is the subroutine used for making sure that usernames only contain alphanumeric characters, using Unity's RegEx pattern for alphanumeric characters. Any username input is checked against this, and the outcome of this validation is returned to the main validation subroutine.

It then declares the subroutines for if the registration was successful (data is sent to external user database), or if the registration was a failure (data not sent to database). The success of this links to success criteria numbers 55, 56 & 57, as a successful registration will include the storage of email, username & password in the external database.

The registration failure subroutine only executes if there is a failure with the transfer of data itself, and is not affected by the previous validations routines.

The final subroutine is for showing feedback to the user using the UI text element 'feedbackText'. This can be used to alert the user of various things,

such as to enter an email in correct format, or to tell them that their registration was successful.

Lines 128 to 133 of
Sign Up Code

```
129 // subroutine attached to the onclick function of the back button
130 0 references
131 public void GoBack()
132 {
133     //uses unity scene manager to load the main menu scene
134     SceneManager.LoadScene("Main Menu");
135 }
```

Green lines of code with a double forward slash (//) preceding them are comments within the code, to ensure that code is understandable and maintainable.

The last few lines of the sign up code are the subroutine for the user to be able to go back to the main menu. This may be needed in many cases, such as if the user accidentally clicks on the 'Sign Up' button instead of the login button, and wants to go back to select a different option.

This links to success criteria number 12, 'Back buttons on all menus' and makes the program more accessible by making it easier to navigate through the different menus.

3.3.3 - Sign Up Menu Iterative Testing

After developing this menu, I have carried out numerous tests on it to see if it functions correctly.

3.3.3.1 - Test Table

Test No.	Test Data	Expected Outcome	Actual Outcome	Success /Failure	Remedial Actions
1	null	User should be alerted to input an email	User was alerted to enter a email	Success	N/A
2	someone.com	User should be alerted to put in an email with the correct format	User was alerted to enter the email in the correct format	Success	N/A
3	someone@example.com	User should not be alerted.	User was not alerted	Success	N/A

4	someone@example.co.uk	User should not be alerted.			
5	testmail@gmail.com and then again with testmail@gmail.com	The user should be alerted that this email has been taken	User was alerted that email is not available	Success	N/A
6	null	User should be alerted to enter a username	User was alerted to enter a username	Success	N/A
7	user??!!\$\$	User should be alerted to only enter a username with alphanumeric characters	User was alerted that usernames should only contain alphanumeric characters	Success	N/A
8	testuser	User should not be alerted.	User was not alerted	Success	N/A
9	test.user	User should be alerted to select a username without any special characters	User was alerted that usernames should only contain alphanumeric characters	Success	N/A
10	thename and then thename again	User should be alerted that this username is not available	User alerted that username is not available	Success	N/A
11	null	User should be alerted to enter a password	User was alerted to enter a password	Success	N/A
12	password	User should not be alerted	User was not alerted	Success	N/A
13	password12345678	User should not be alerted	User was not alerted	Success	N/A
14	typicalpass	User should not be alerted	User was not alerted	Success	N/A
15	short	User should be alerted to pick a password between 8	User was told to ensure that password	Success	N/A

		and 16 characters (inclusive)	is between 8 and 16 characters		
16	longerpassword1234	User should be alerted to pick a password between 8 and 16 characters (inclusive)	User was told to ensure that password is between 8 and 16 characters	Success	N/A
17	null	User should be alerted to input a confirm password	User was alerted to ensure that passwords match	Failure	Need to add code to see if confirm password field is empty
18	typicalpass [identical to previously entered password]	User should not be alerted		Success	N/A
19	notthesamepass	User should be alerted to make the confirm password the same as	User was alerted to ensure that passwords matched	Success	N/A

3.3.3.2 - Test Evidence

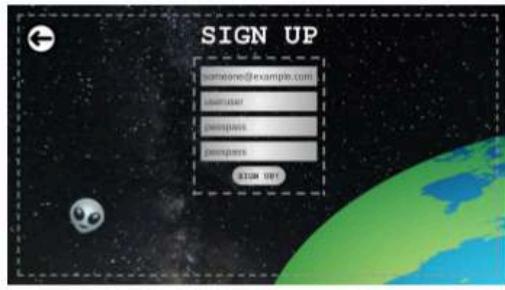
Test No. 1



Test No. 2



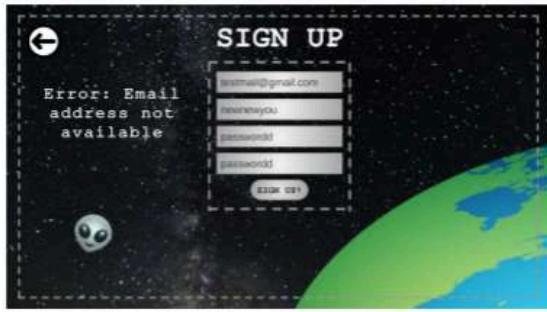
Test No. 3



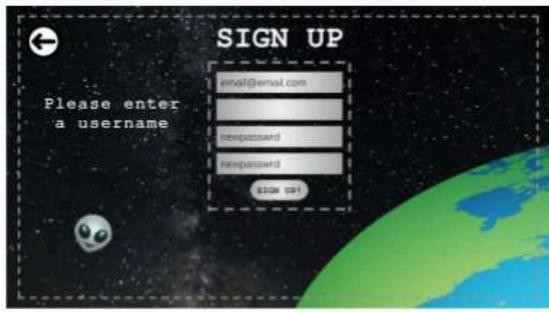
Test No. 4



Test No. 5



Test No. 6



Test No. 7



Test No. 8



Test No. 9

SIGN UP

Username can only contain alphanumeric characters.

email@email.com
test_user1
newpassword
newpassword

SIGN UP!

Test No. 10

SIGN UP

Error: Username not available

email@email.com
username
password123
password123

SIGN UP!

Test No. 11

SIGN UP

Please enter a password

email@email.com
im11234
password123

SIGN UP!

Test No. 12

SIGN UP

emailuser2@newmail.com
username2
password
password

SIGN UP!

Test No. 13

SIGN UP

email@email.com
username123
password12345678
password12345678

SIGN UP!

Test No. 14

SIGN UP

email@yahoo.co.uk
username123
typicalpass
typicalpass

SIGN UP!

Test No. 15

SIGN UP

Ensure password is between 8 and 16 characters

email@email.com
username123
short
short

SIGN UP!

Test No. 16

SIGN UP

Ensure password is between 8 and 16 characters

email@email.com
username123
longpassword1234
longpassword1234

SIGN UP!

Test No. 17



Test No. 18



Test No. 19



3.3.4 - Review of Sign Up Menu Development

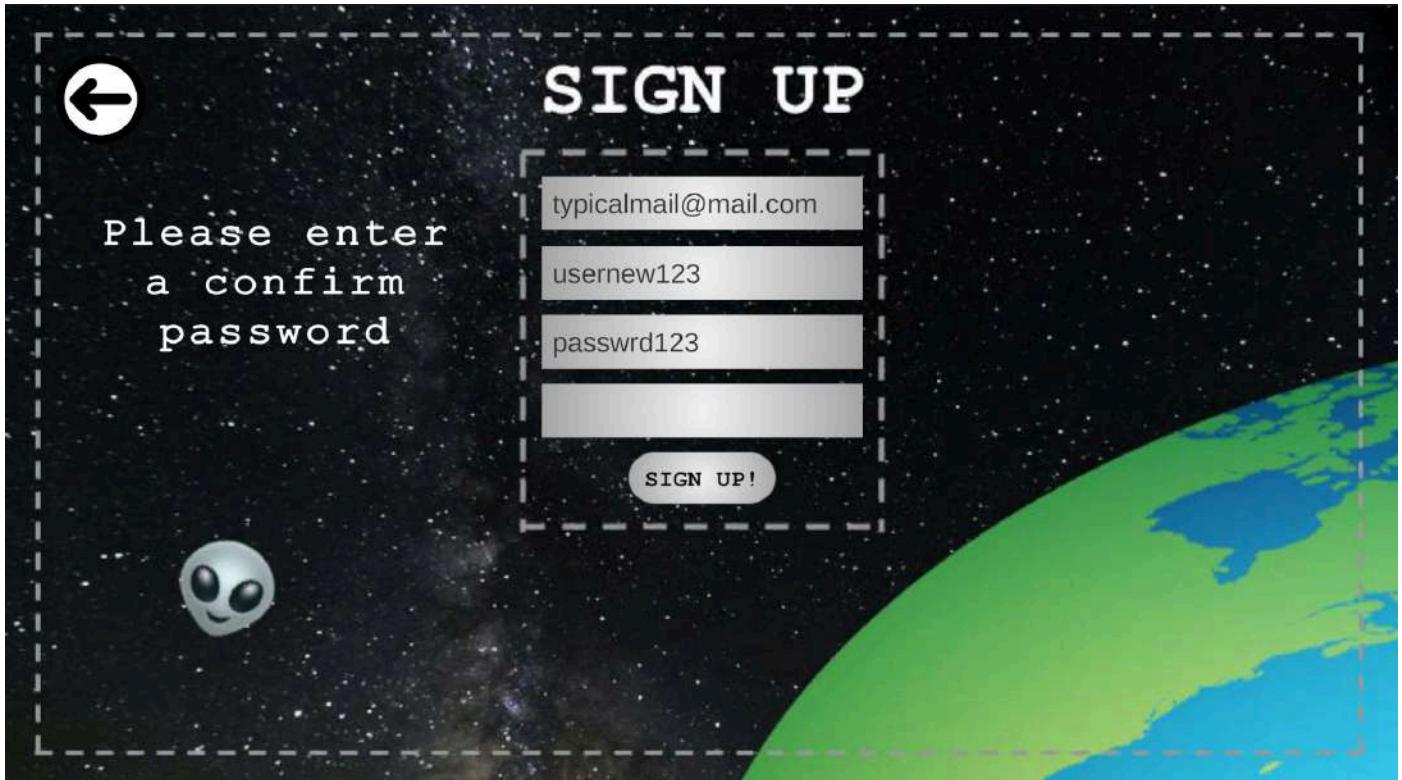
Out of all of the tests outlined in [2.11.1 - Sign Up Module Testing](#) there was 1 failed test within for my 'Sign Up Menu', test number 17, a test to see if a user will be alerted of a null confirm password field.

- This must be an explicit message, as the message that appeared ('Please ensure that passwords match.') is vague and doesn't directly address the issue, the fact that the field is null.

To rectify this, I will be adding code (and informative comments) that alerts the user of the 'confirm password' field being null before I move onto the next stage of development, the login menu. The code I have added is as follows:

```
51 //Action taken based on testing: Checks if there is an input in the 'confirm password' field
52 if (string.IsNullOrEmpty(pass2Confirm))
53 {
54     feedbackText.text = "Please enter a confirm password";
55     return;
56 }
```

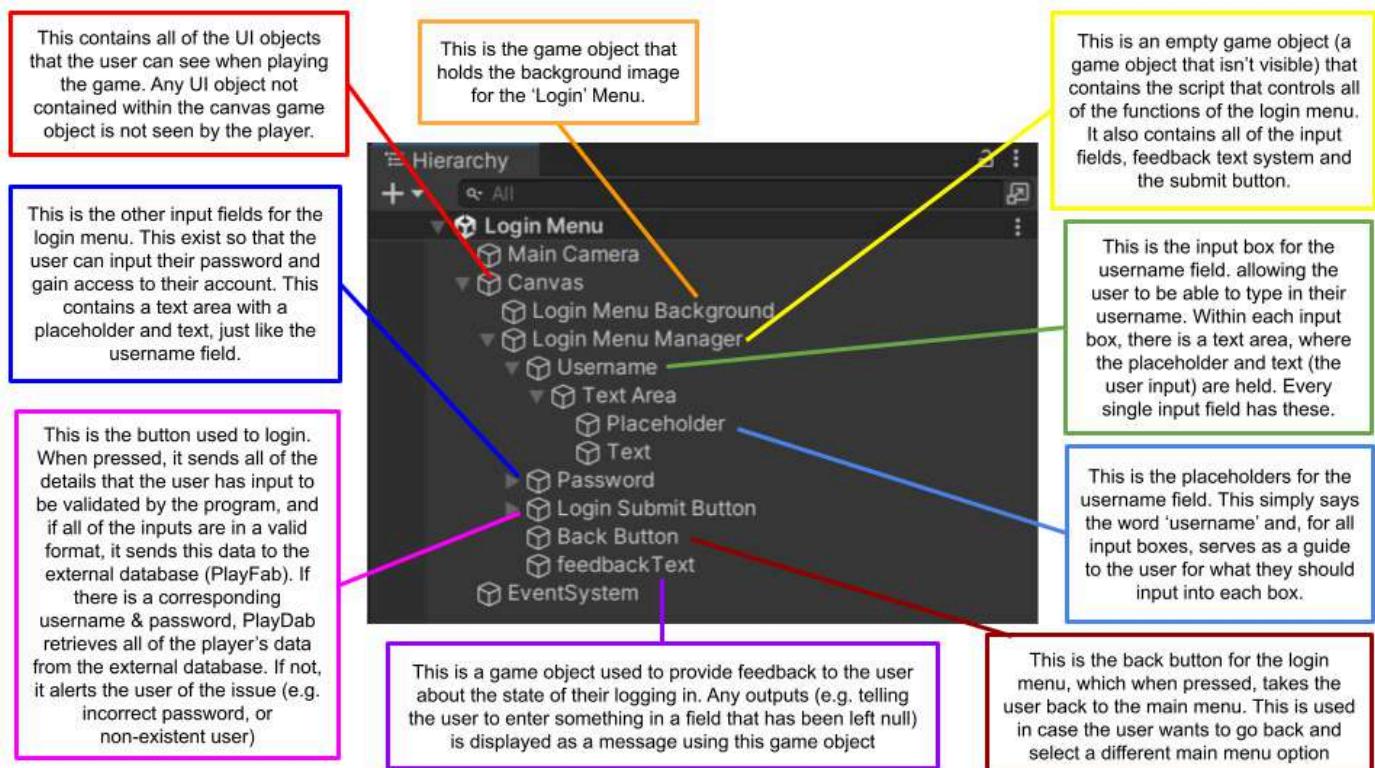
And I have then tested it using the same test data as in Test 17, and the result was as follows:



There is now an issue-specific message for when the confirm password field is not filled out. This has rectified the issues found in the Sign Up menu.

3.4 – Login Menu Development

3.4.1 – Login Menu Game Objects



3.4.2 - Login Menu Code

Lines 1 to 9 of
Login Code

```
1 //importing libraries to use different functions in my game
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.UI;
6 using PlayFab;
7 using PlayFab.ClientModels;
8 using TMPro;
9 using UnityEngine.SceneManagement;
10
```

Lines 1 to 9 of my Login Menu code are declaring the libraries to be used in-game. This code allows functions such as scene management function, the use of TextMeshPro input fields & PlayFab integration to be used within my project.

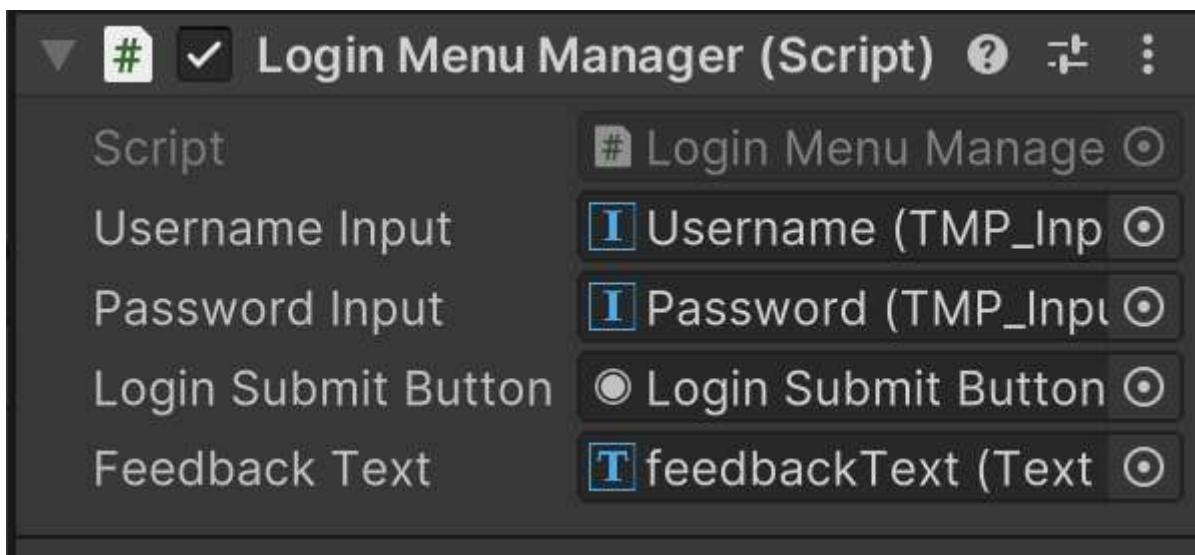
By importing these libraries, I am able to fulfill success criteria number 12 ('using UnityEngine.SceneManagement' for the back button), 15 ('using TMPro' for login system input fields' & 52 to 62 ('using PlayFab' & 'using PlayFab.ClientModels' for being able to retrieve and send data to external database).

Lines 11 to 28 of
Login Code

```
11 public class LoginMenuManager : MonoBehaviour
12 {
13     // initialises the UI elements for user inputs (username & password), feedback & login submit button
14     public TMP_InputField usernameInput;
15     public TMP_InputField passwordInput;
16     public Button loginSubmitButton;    // Button for submitting the login request
17     public TMP_Text feedbackText;      // text field to display feedback messages
18
19     private string playFabTitleId = "45CD5"; // The PlayFab Title ID for AstroSprint, establishing the connection to the external database
20
21     private void Start()
22     {
23         // Set the PlayFab Title ID for the current session.
24         PlayFabSettings.staticSettings.TitleId = playFabTitleId;
25
26         // Add a listener, something that waits for the button to be clicked, and then calls the login method onclick
27         loginSubmitButton.onClick.AddListener(Login);
28     }
29 }
```

The overall purpose of these lines of code is to initialise the user inputs for validation, and to set up the playfab request for when the user presses submit.

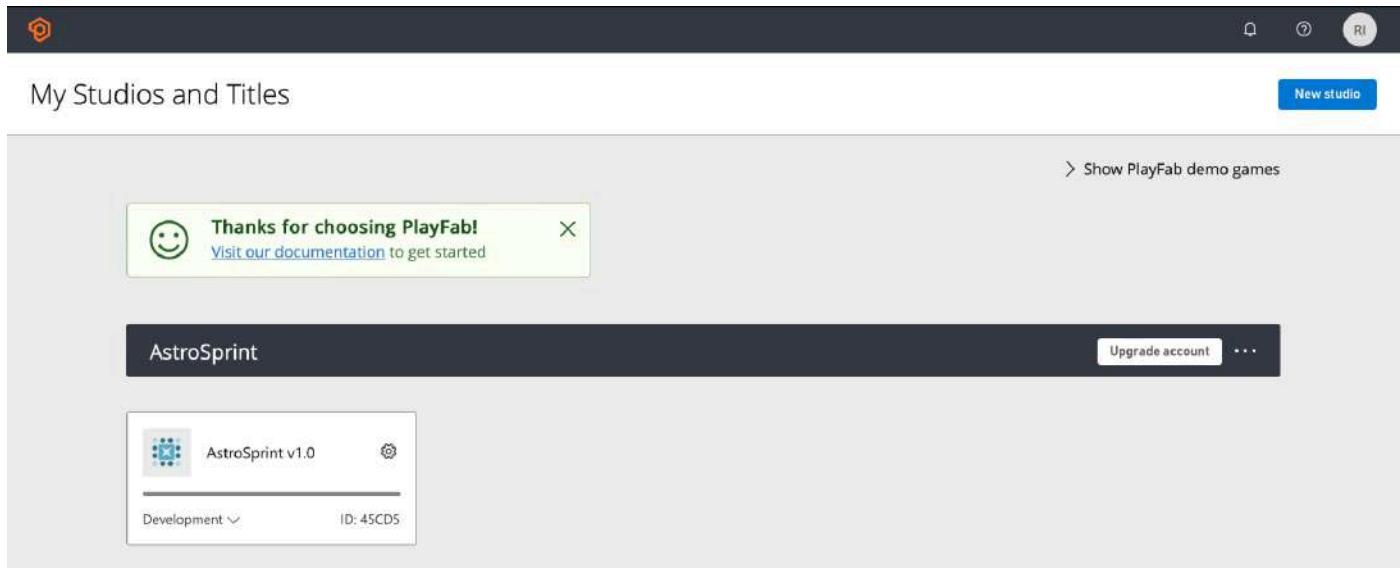
The first section initialises the UI element for the user inputs. These inputs are the username & password fields. This code allows for the input fields, feedback text & a submit button to send all the data to be assigned to variables later on in the script, by dragging them into the corresponding box in the inspector (pictured below)



This links to success criteria number 15, 'Login System', as this is what links the user input in the Unity scene to the external C# script in Visual Studio, where all the validation takes place.

The next section (Section 2) of this code declares the title ID for AstroSprint in the script. This is essential for success criteria number 15

(and later on 52 to 62), as it is the title ID that allows me to link the external database to my game. In the login system, the title ID enables the PlayFab request to go to the correct game studio.



As shown in the image above, '45CD5' is the corresponding ID for the first prototype of my game, AstroSprint v1.0.

The last section (Section 3) of my code is the subroutine that is called the moment the scene is opened. It has 2 functions, setting the PlayFab title ID as the one in the declared variable in the previous section & attaches a listener to the login button. It is necessary to set the PlayFab title ID as the one previously declared in the variable 'playFabTitleID' so that all of the PlayFab API calls are going to the correct title, 'AstroSprint v1.0'. It is also necessary to add a listener, a function that waits for the button to be clocked before calling the Login subroutine, to the login submit button so that when it is clicked, it can call the login subroutine and begin the validation process. This links to success criteria number 15, as the listener is what allows for the process to start & the login request to be sent to PlayFab for authentication.

Lines 30 to 66 of
Login Code

```

38 // handles the login process after the button is clicked, including validation and PlayFab API calls
39
40 // Check if the username field is empty and alerts user to enter username if it is
41 if (string.IsNullOrEmpty(usernameInput.text))
42 {
43     feedbackText.text = "Please enter a username";
44     return;
45 }
46
47 // creates a PlayFab login request for the inputted username & password
48 var request = new LoginWithPlayFabRequest
49 {
50     Username = usernameInput.text,
51     Password = passwordInput.text
52 };
53
54 // calls PlayFab's login API and handle the success or failure of the login
55 PlayFabClientAPI.LoginWithPlayFab(request, OnLoginSuccess, OnLoginFailure);
56
57
58 // subroutine for successful login
59 private void OnLoginSuccess(LoginResult result)
60 {
61     // Display a success message with the username.
62     feedbackText.text = "Login successful! Welcome " + usernameInput.text;
63
64     // Loads the Play Menu after login
65     SceneManager.LoadScene("Play Menu");
66 }

```

The first section (Section 1) of this code begins by defining the login subroutine; all of the login validation subroutines are within this larger subroutine. This then checks if the username input field is empty, and if it is, the user is alerted of this. The next section (Section 2) is a subroutine within the login subroutine for checking if the password field is null. This is necessary for both usernames and passwords, so that data is actually there is data to be sent to PlayFab.

The next section (Section 3) is creating & sending a new request to PlayFab to login. It is this process in which PlayFab is later able to check whether the user exists, and if the password entered is the one that corresponds to the username entered. This is a major part of success criteria 15 ('Login System'), as this is where (after validation) the user is authenticated by the system, and allowed into the game.

The last section (Section 4) is the subroutine called if the PlayFab login is successful. This essentially means that the username exists and the password entered was the corresponding password for that user. This displays a message to the user that their login was successful, and welcomes them. Using the scene management function in Unity, it then loads the play menu.

All of these sections link to success criteria number 15, as they make up a major aspect of the validation process and the login system as a whole.

Lines 68 to 80 of
Login Code

```

68     // subroutine for login failure
69     private void OnLoginFailure(PlayFabError error)
70     {
71         // displays an error message with details from PlayFab
72         feedbackText.text = "Login failed: " + error.ErrorMessage;
73     }
74
75     //allows the user to go back to the main menu
76     public void GoBack()
77     {
78         SceneManager.LoadScene("Main Menu");
79     }
80 }
```

The final lines of the login menu are for the case of a login failure, and a scene management function.

The first section (Section 1) first describes the subroutine that is called if the login fails. This retrieves the error from PlayFab (`error.ErrorMessage`), concatenates it with 'Login failed: ' and displays it to the user through the `feedbackText` game object

Section 2 of this code is for the back button on the login menu, linking to success criteria number 12, as it is this subroutine that utilises the scene manager's 'LoadScene' function in order to (when the back button is clicked) go back to the main menu. This is beneficial as it allows smooth navigation throughout the menus in the game.

3.4.3 - Login Menu Iterative Testing

3.4.3.1 - Test Table

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/ Failure	Remedial Actions
Username Input Box	20	null	User should be alerted to input a username	User was told to enter a username	Success	N/A
	21	fakeuser	User should be alerted that username is not valid	User was alerted that username was not found	Success	N/A
	22	testuser	User should not be alerted	User was not alerted	Success	N/A
Password Input Box	23	null	User should be alerted to enter a password	User was told to enter a password	Success	N/A

d Input Box			alerted to enter a password	to enter a password		
	24	invalidpass	User should be alerted that their password is not valid	User was alerted that their username or password was invalid	Success	N/A
	25	typicalpass	User should not be alerted	User was not alerted	Success	N/A

3.4.3.2 - Test Evidence

Test No. 20



Test No. 21



Test No. 22



Test No. 23



Test No. 24



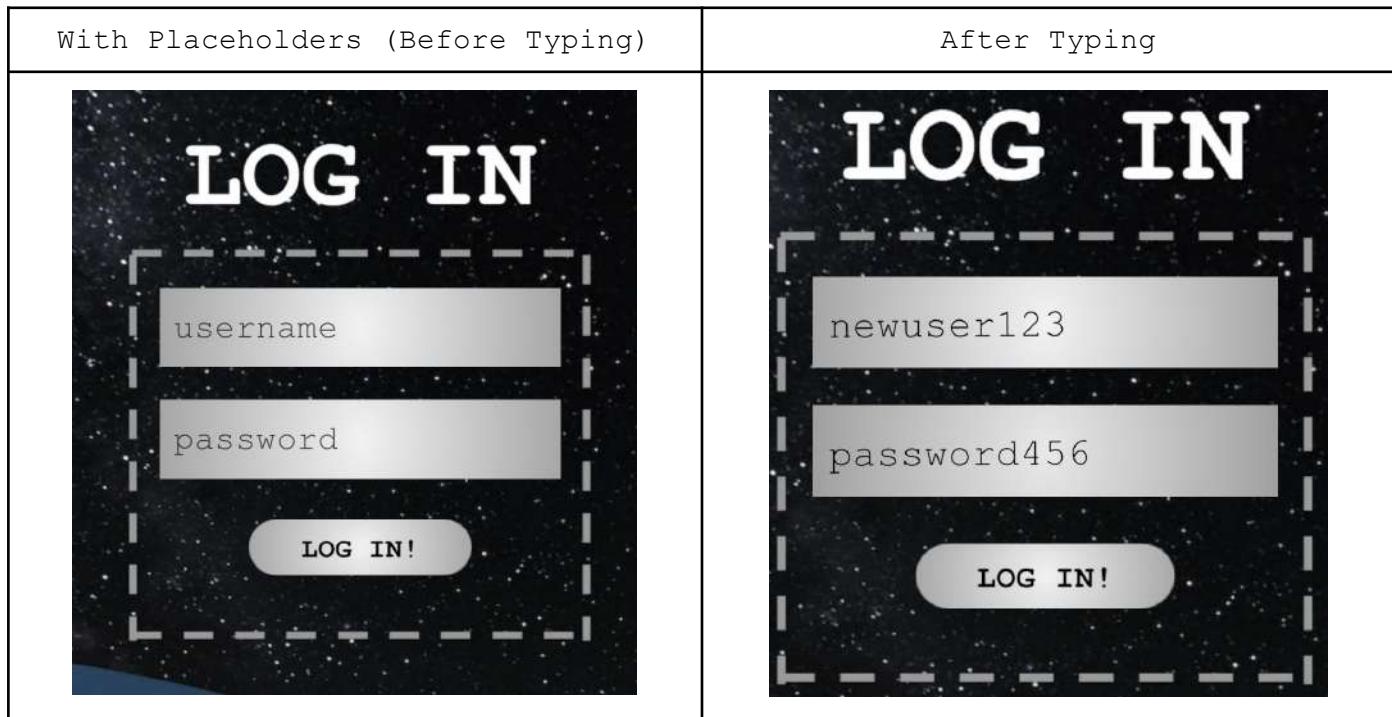
Test No. 25



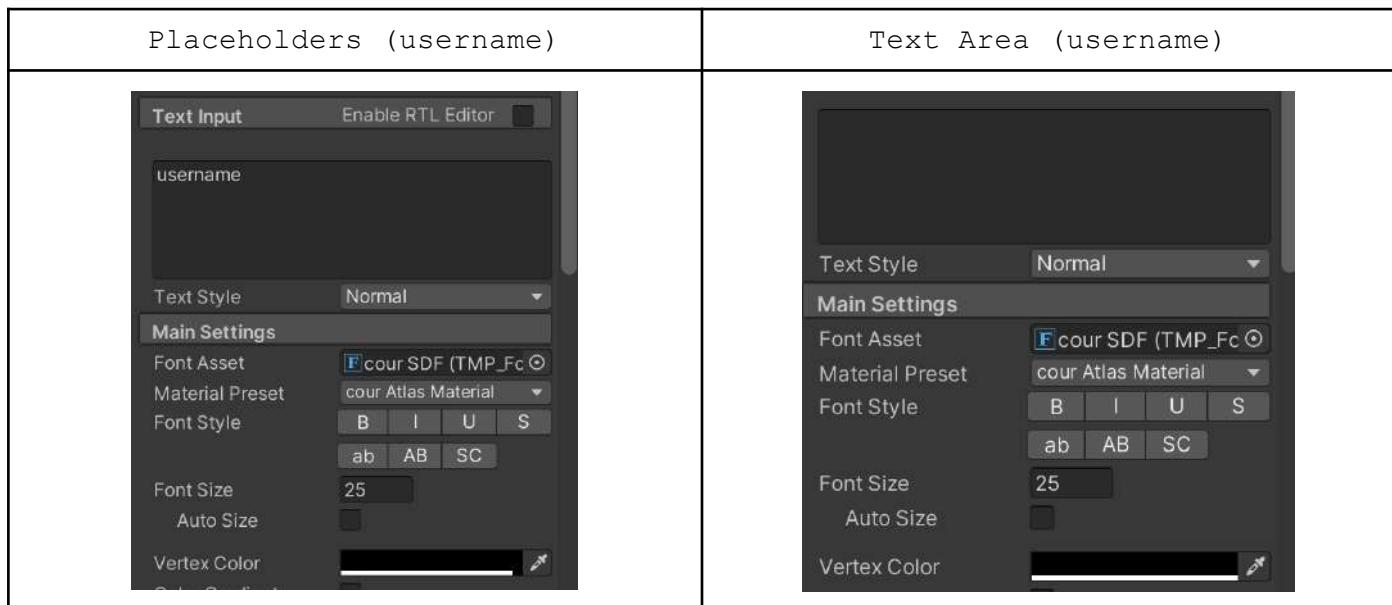
3.4.4 - Review of Login Menu Development

There were no failed tests during my login menu development. However, I described in my analysis & usability features how I wanted my game to be accessible to the visually impaired, and on testing, it has come to my attention that the input box text may be difficult to read.

When typing in a username and password, it looks like the images as follows:



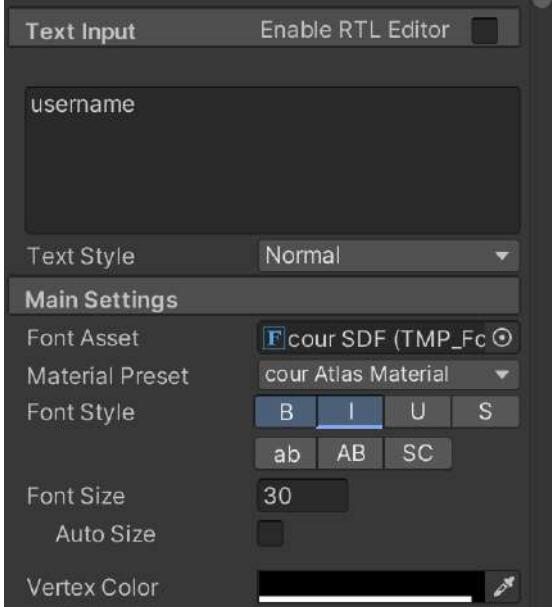
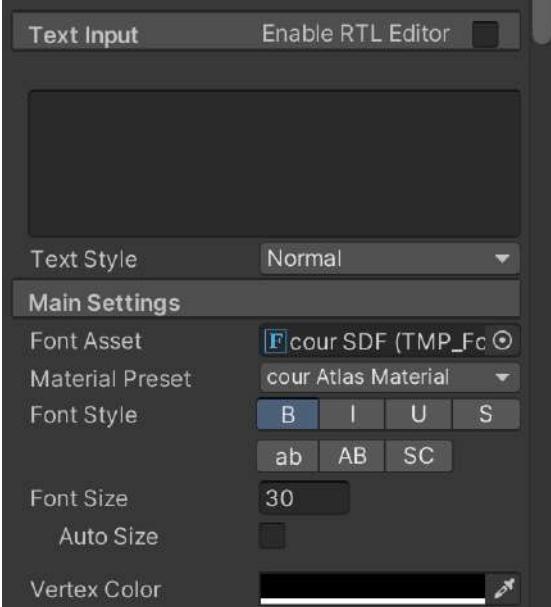
Not only does the text colour slightly match the background in both, but the text, being very thin, is very difficult to read. There is also no distinction between the formatting of the placeholder text & the input text, as shown in their respective inspectors:



Therefore, for better accessibility to those with worse eyesight, I have decided to make both texts in bold, and of a larger font size.

- Additionally, for the placeholders, I have decided to make them in italics to help differentiate easily between the placeholders and what the user has entered.

My changes are as follows:

Placeholders	Text Area
	

Placeholders:

- Font size increased from 25 to 30
- Text in bold
- Text in italics

Text Area:

- Font size increased from 25 to 30
- Text in bold

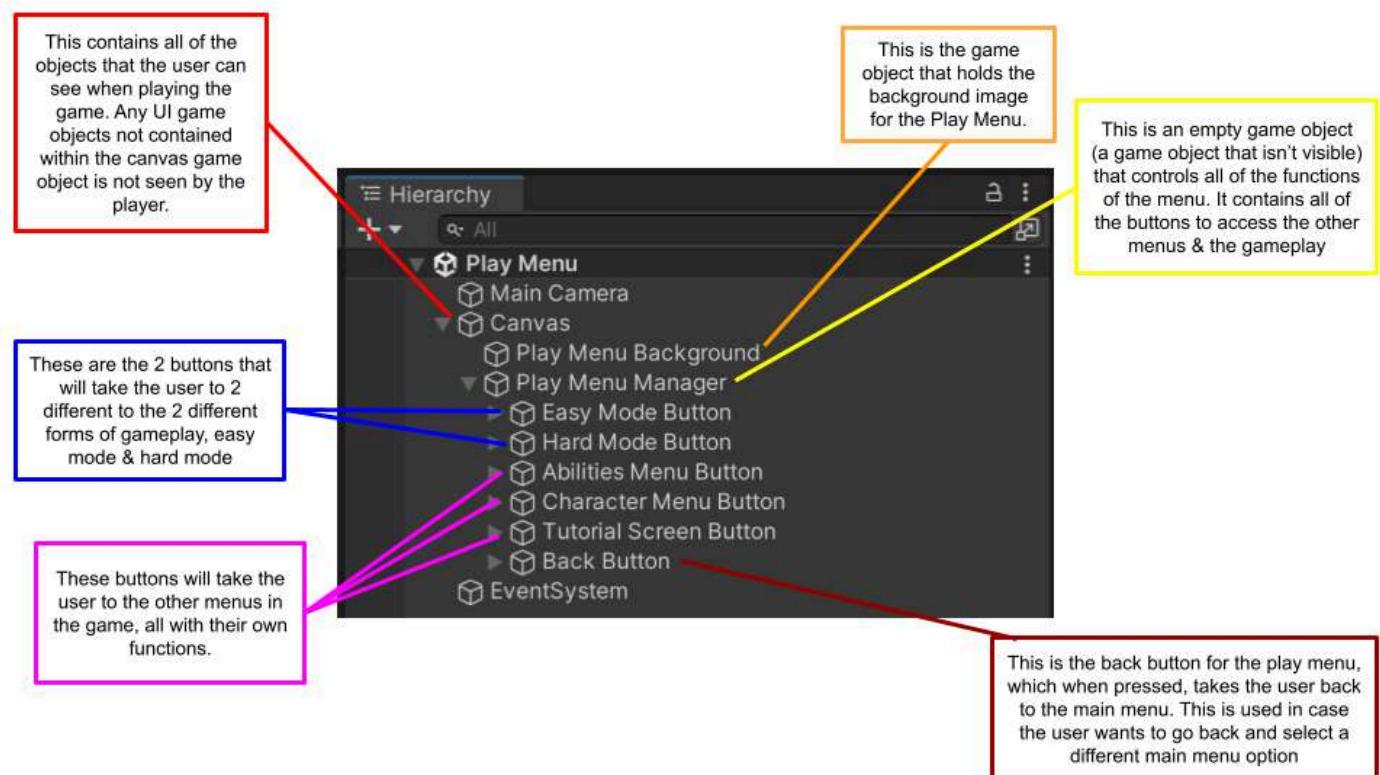
4 - Second Iteration Development (AstroSprint v2.0)

4.1 - AstroSprint v2.0 Summary

Below is the development of the second iteration of my code, which includes the previously developed menus (Main Menu, Login Menu & Sign Up Menu), as well as 4 additional menus: Play Menu, Character Menu & Abilities Menu. All together, these make up AstroSprint v2.0.

4.2 - Play Menu Development

4.2.1 - Play Menu Game Objects



4.2.2 - Play Menu Code

Lines 7 to 31 of
Play Menu Code

```
7  public class PlayMenuManager : MonoBehaviour
8  {
9      //takes user to abilities menu
10     0 references
11     public void GoToAbilities()
12     {
13         SceneManager.LoadScene("Abilities Menu");
14     }
15     //takes user to characters menu
16     0 references
17     public void GoToCharacters()
18     {
19         SceneManager.LoadScene("Character Menu");
20     }
21     //takes user to easy mode gameplay
22     0 references
23     public void GoToEasyMode()
24     {
25         SceneManager.LoadScene("Easy Mode");
26     }
27     //takes user to hard mode gameplay
28     0 references
29     public void GoToHardMode()
30     {
31         SceneManager.LoadScene("Hard Mode");
32     }
33 }
```

The first section of this code is for the on-click of the abilities menu button. This (as labelled by the comments in green) takes the user to the abilities menu. The next section has a similar function, taking the user to the characters menu.

Sections 3 & 4 entail taking the user to the gameplay. Section 3 shows the subroutine for the onclick of the 'Easy Mode' button, which takes the user to the easy mode gameplay & section 4 takes the user to the hard mode gameplay.

All of these buttons and their subroutines are necessary for smooth navigation between menus, and for the user to be able to see all of the options for the game and move between them. This links to success criteria number 2 ('Clearly labelled menus, with options within them being clear') as these buttons on the Play Menu clearly indicate what their functions are, and are clear.

Lines 33 to 43 of
Play Menu Code

```
33 // takes user to tutorial scene
34 0 references
35 public void GoToTutorial()
36 {
37     SceneManager.LoadScene("Tutorial Screen")
38
39 //allows user to use back button to go back to main menu
40 0 references
41 public void GoBack()
42 {
43     SceneManager.LoadScene("Main Menu");
```

The last few lines of the Play Menu code include the navigation buttons to the tutorial, and the back button. Section 1 is the code that takes the user to the tutorial & section 2 is the code that takes the user back to the main menu.

The fact that tutorial is labelled as a 'Screen' and not a 'menu' is essential, as in-game, it will not be interactable like the other menus are, but will only be a screen that displays the controls of the game. This lack of interactivity (apart from the back button) means that this is not a menu, but a screen. This is related to success criteria number 2 ('Clearly labelled menus, with options within them being clear') as the differentiation between calling it a screen and not a menu makes the user aware that it is not a menu.

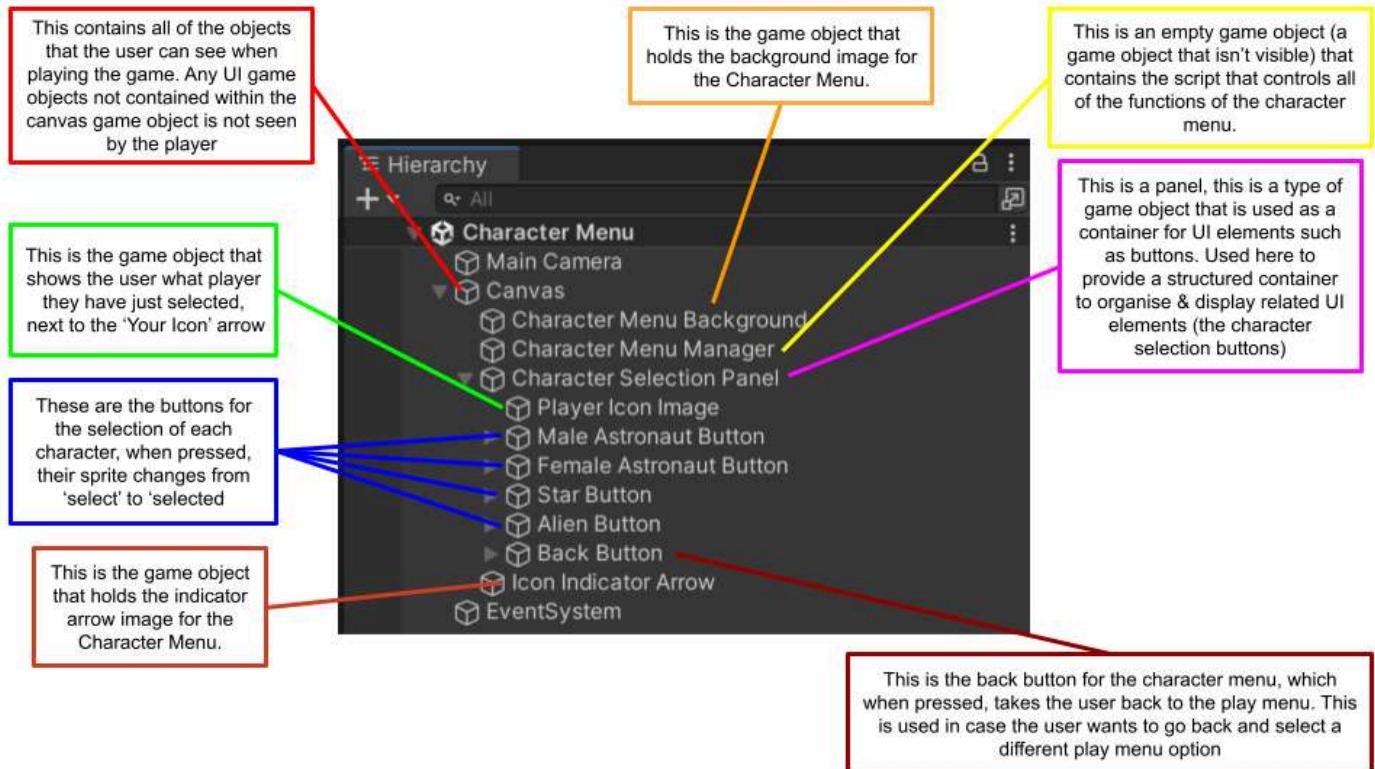
4.2.3 - Review of Play Menu Development

I believe that my code for my play menu will be successful, while being very concise, which is beneficial as it saves memory, and overall enables the game to run faster.

I am unable to test this menu until the others have been developed, therefore, I will be testing the Play Menu in the post development testing, so that I can see if it correctly leads to the other menus.

4.3 - Character Menu Development

4.3.1 - Character Menu Game Objects



4.3.2 - Character Menu Code

Lines 9 to 27 of Character Menu Code

```

9  public class CharacterManager : MonoBehaviour
10 {
11     // 4 references
12     public Image playerIconImage; // image to show user what their selected icon is
13     public Sprite maleAstronautIcon; // sprite image for male astronaut icon
14     public Sprite femaleAstronautIcon; // sprite image for female astronaut icon
15     public Sprite starIcon; // image sprite for star icon
16     public Sprite alienIcon; // image sprite for alien icon
17
18     // 0 references
19     private void Start()
20     {
21         // retrieve the previously selected character from PlayFab
22         FetchCharacterFromPlayFab();
23     }
24
25     // retrieves character selection from PlayFab, and calls subroutines based on the result
26     PlayFabClientAPI.GetUserdata(new GetUserdataRequest(), OnDataReceived, OnError);
27 }
```

The first section of this code is the declaration of all of the sprites (the icons which will be used in-game) and the player icon image (image game object that shows the user the character they selected). The declaration of

sprites is to enable the images to be attached to the script in the inspector, allowing the script to have access to the character images in order to show it on the player icon image. This links to success criteria number 8, as the declaration of sprites allows them to be displayed in the character menuUI.

The next section of this code is the code that is run at the start of the program. In this section, the subroutine 'FetchCharacterFromPlayFab()' is called, which retrieves the most recently selected character from PlayFab for the user.

The last section of this code creates a new PlayFab data request, which enables the retrieval of data from the external database to be used in the game. In this case the data retrieved is the name of the character previously selected by the player. This section and the previous section links to success criteria number 54, storing the selected character, as this is the use of the storage of the character.

Lines 29 to 50 of Character Menu Code

```
1 reference
29 private void OnDataReceived(GetUserDataResult result)
30 {
31     // retrieves the character name from PlayFab (if it is there)
32     string selectedCharacter = "MaleAstronaut"; // defaults to male astronaut character if none is selected
33
34     // checks that the "SelectedCharacter" key is present in PlayFab before attempting to retrieve data from it
35     if (result.Data != null && result.Data.ContainsKey("SelectedCharacter"))
36     {
37         selectedCharacter = result.Data["SelectedCharacter"].Value;
38     }
39
40     // update the player icon image based on the selected character
41     UpdatePlayerIcon(selectedCharacter);
42 }
```

1


```
2 references
44 private void OnError(PlayFabError error)
45 {
46     // debug log of any errors
47     Debug.LogError("Error fetching data from PlayFab: " + error.GenerateErrorReport());
48     UpdatePlayerIcon("MaleAstronaut"); // Defaults to MaleAstronaut if there's an error
49 }
```

2

The next section of this code is the subroutine that runs if the character selection data is successfully received from PlayFab. It saves the retrieved data to a variable named "selectedCharacter", and then uses that variable to call another subroutine which updates the player icon. The calling of this subroutine links to success criteria number 8, as this is the subroutine that is used to display the character selected on the character menu UI. This is a major aspect of the character menu UI.

The last section of this code is the subroutine that runs when there is an error in retrieving the name of the currently selected character from PlayFab. In this case, the character icon is defaulted to the 'Male

Astronaut' icon. This is to ensure that, regardless of the outcome of the PlayFab request, the user always has an icon to display. This links to success criteria number 8, as the existence of a default icon in the event of an error ensures that the gameplay is still seamless.

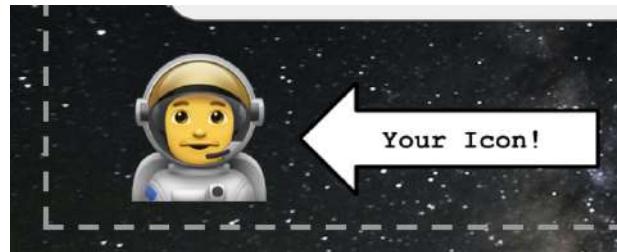
Lines 51 to 82 of Character
Menu Code

```
51. private void UpdatePlayerIcon(string characterName)
52. {
53.     switch (characterName)
54.     {
55.         // changes the image of the player icon image depending on what character name is retrieved
56.         case "MaleAstronaut":
57.             playerIconImage.sprite = maleAstronautIcon;
58.             break;
59.         case "FemaleAstronaut":
60.             playerIconImage.sprite = femaleAstronautIcon;
61.             break;
62.         case "Star":
63.             playerIconImage.sprite = starIcon;
64.             break;
65.         case "Alien":
66.             playerIconImage.sprite = alienIcon;
67.             break;
68.         default:
69.             // debug log for any errors
70.             Debug.LogWarning("Unknown character selected: " + characterName);
71.             break;
72.     }
73. }
74.
75. public void OnIconSelected(string characterName)
76. {
77.     // saves the selected character to PlayFab
78.     SaveCharacterToPlayFab(characterName);
79.
80.     // updates the player icon based on retrieved
81.     UpdatePlayerIcon(characterName);
82. }
```

1

2

The first section of this code is the subroutine to update the player icon image shown to the user (shown on the right). The subroutine accepts the parameter 'characterName', which is passed in when the subroutine is previously called. It compares this variable to the names of all the different icons, and then changes the sprite accordingly. This change updates the UI to show the user what character they have selected. This links to success criteria number 8, as this is the section of code that directly updates the player icon image, showing the user what character they chose on the character menu UI.



The default case for this switch statement is a message in the debug log, to ensure maintainability and ease of testing for this code. This is useful in testing because it will show any errors that have been made during the development of the character selection.

Lines 84 to 109 of Character Menu Code

```
84 // creates a new PlayFab request to save new character selections
85     reference
86     private void SaveCharacterToPlayFab(string characterName)
87     {
88         var request = new UpdateUserDataRequest
89         {
90             Data = new Dictionary<string, string>
91             {
92                 { "SelectedCharacter", characterName }
93             }
94         };
95
96         // Save the selected character to PlayFab
97         PlayFabClientAPI.UpdateUserData(request, OnDataSaved, OnError);
98     }
99
100    reference
101    private void OnDataSaved(UpdateUserDataResult result)
102    {
103        Debug.Log("Character selection saved to PlayFab.");
104
105    }
106
107    0 references
108    public void GoBack()
109    {
110        // Go back to the play menu
111        SceneManager.LoadScene("Play Menu");
112    }
113
```

The first section of this code is the code used to save the character selection to PlayFab after it has been displayed to the user. Using the variable 'characterName', this subroutine is able to save the character selection to PlayFab, to be accessed the next time the user logs in. This links to success criteria number 54, as this is the code that directly stores the name of the selected character to PlayFab.

The next section of this code is for testing and maintainability only, as it just sends a debug log message to the console about the success of the character selection being saved to PlayFab.

The final section of this code is the code that enables the back button to take the user back to the previous screen, the 'Play Menu'. This links to success criteria number 12, having back buttons on all menus, as it enables the back button on the character menu, when clicked, to return to the Play Menu.

4.3.3 - Character Menu Iterative Testing

4.3.3.1 – Test Table

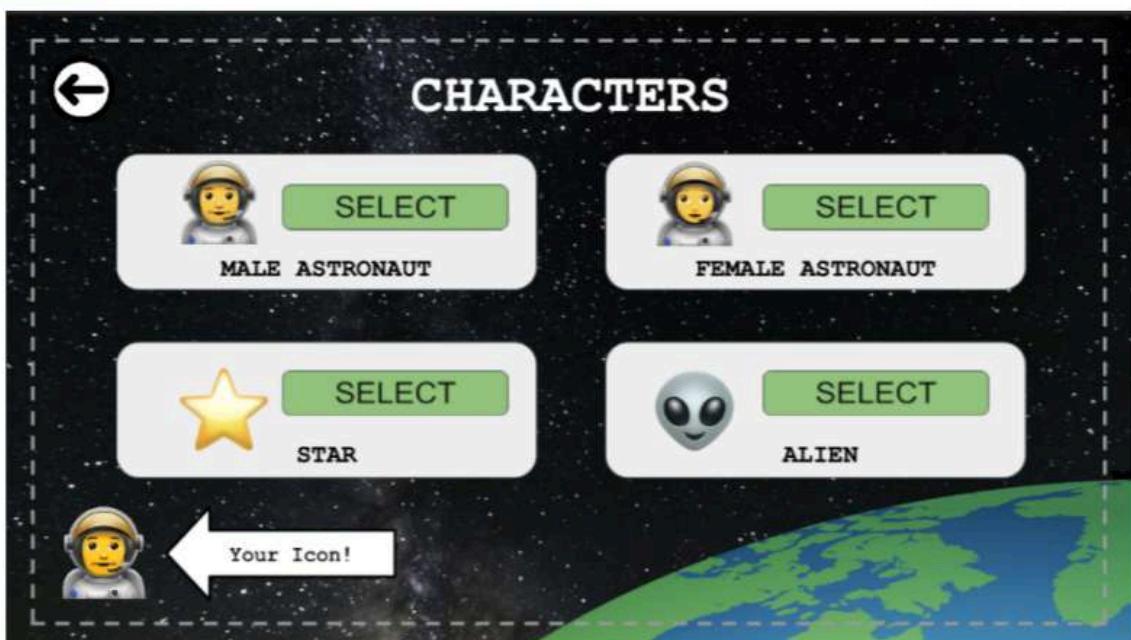
Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/Failure	Remedial Actions
Initial Character	26	N/A	Male astronaut appears next to	Male astronaut	Success	N/A

Selection Display			the 'Your Icon' arrow	icon appeared		
New Character Selection	27	Click on 'Select' for female astronaut	Female astronaut button changes from 'select' to 'selected'. Character next to 'your icon' changes to female astronaut	Female astronaut selection button changes from 'select' to 'selected' Debug log says 'Unknown character: Female Astronaut' and icon stays as male astronaut	Failure	Change information in the inspector for the selection of the female astronaut
	28	Click on 'Select' for male astronaut	Male astronaut button changes from 'select' to 'selected'. Character next to 'your icon' changes to male astronaut	Male astronaut selection button changes from 'select' to 'selected' Debug log says 'Unknown character: Star' and icon stays as male astronaut	Failure	Change information in the inspector for the selection of the male astronaut
	29	Click on 'Select' for the star	Star button changes from 'select' to 'selected'. Character next to 'your icon' changes to star	Star selections button changes from 'select' to 'selected' Debug log says 'Unknown character: Alien' and icon stays as male astronaut	Failure	Change information in the inspector for the selection of the star
	30	Click on 'Select' for the alien	Alien button changes from 'select' to 'selected.'	Alien selection button changes from 'select' to	Failure	Change information in the inspector for the selection of

		Character next to 'your icon' changes to alien	'selected' Debug log says 'Unknown character: Male Astronaut' and icon stays as male astronaut		the alien
31	Click on random part of the screen with no button	Menu remains in the same state that it was before the click happened.	Nothing happened	Success	N/A

4.3.3.2 - Test Evidence

Test No. 26



Test No. 27



Test No. 28

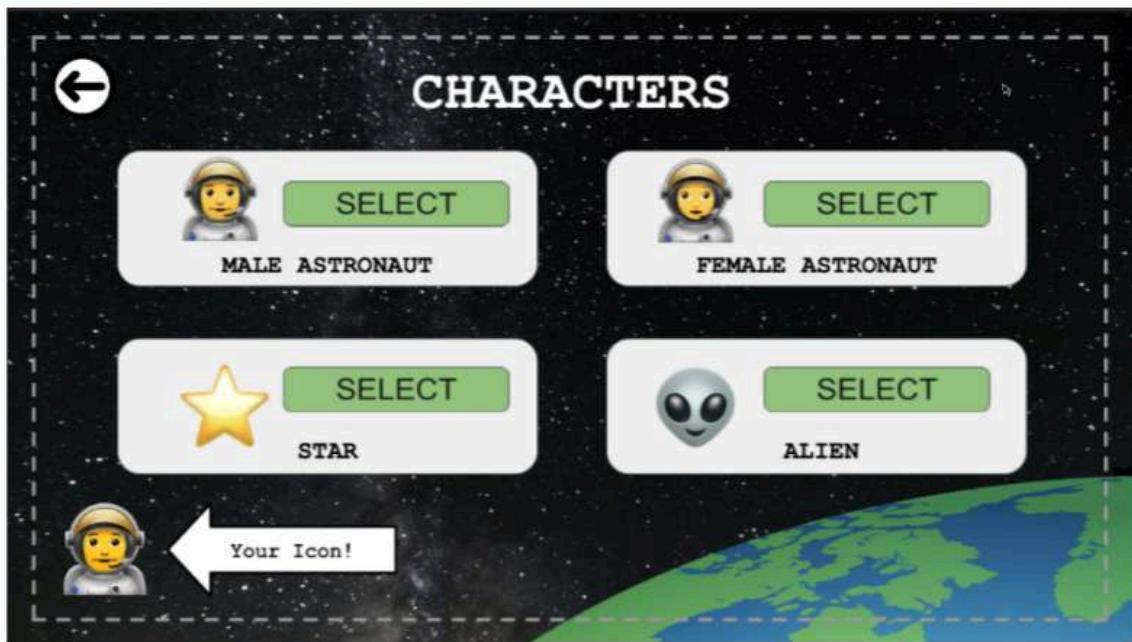


Test No. 29



Test No. 30





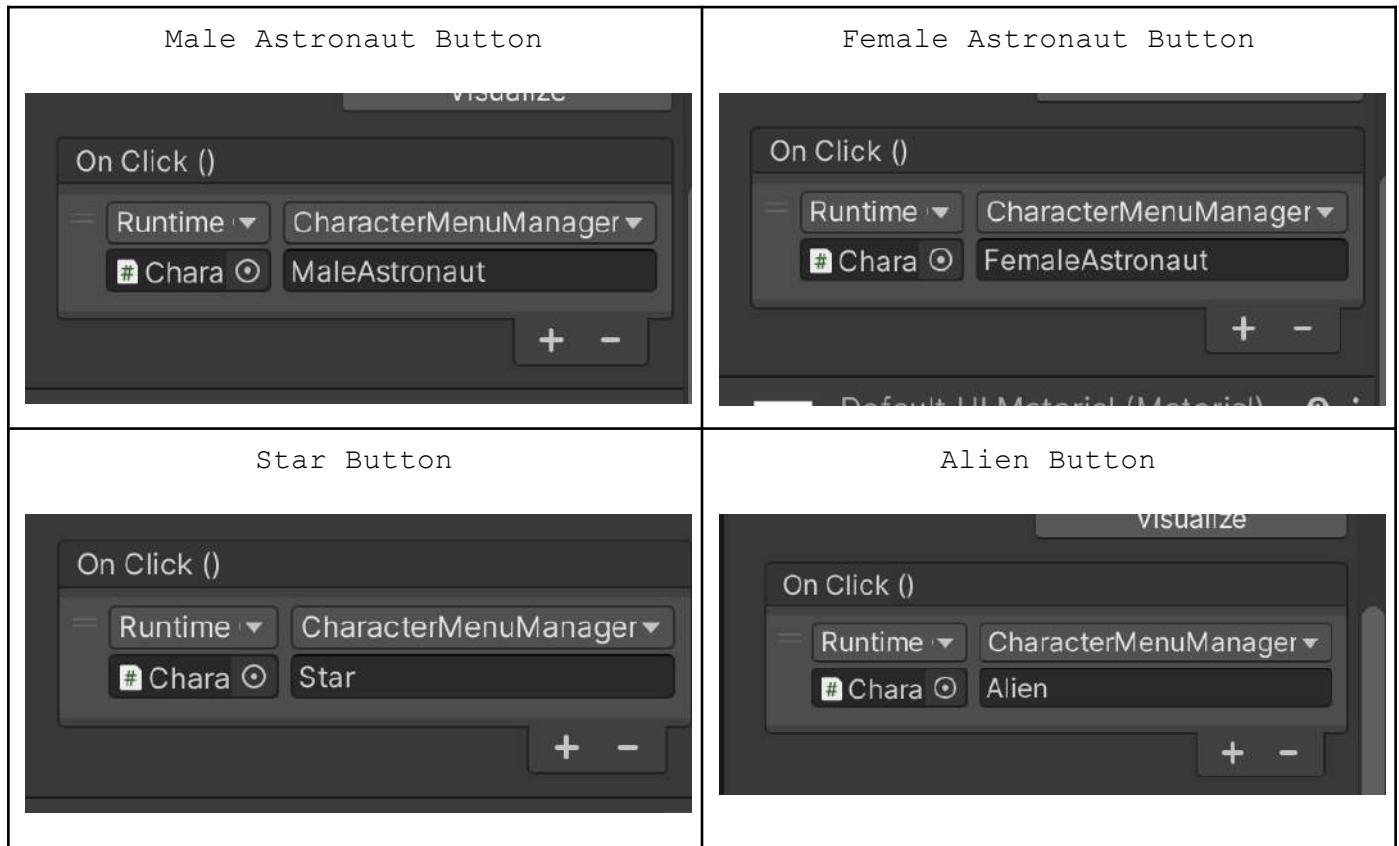
4.3.4 - Review of Character Menu Development

There are remedial actions to take in order to rectify the 4 failed tests that I have carried out. After looking at the code (which had no errors), I found that what I had written in the inspector for each power up button's corresponding name, I have seen that I have inputted the names incorrectly:

<p>Male Astronaut Button</p> <pre>On Click () = Runtime ▾ CharacterManager # Chara ⚪ "Alien"</pre>	<p>Female Astronaut Button</p> <pre>On Click () = Runtime ▾ CharacterManager # Chara ⚪ "FemaleAstronaut"</pre>
<p>Star Button</p> <pre>On Click () = Runtime ▾ CharacterManager # Chara ⚪ "Star"</pre>	<p>Alien Button</p> <pre>On Click () = Runtime ▾ CharacterManager # Chara ⚪ "MaleAstronaut"</pre>

This caused an issue, as the program (in the 'if' statement) was looking for the character names including the quotation marks, when it should have been without.

Therefore, I removed them, as shown below:



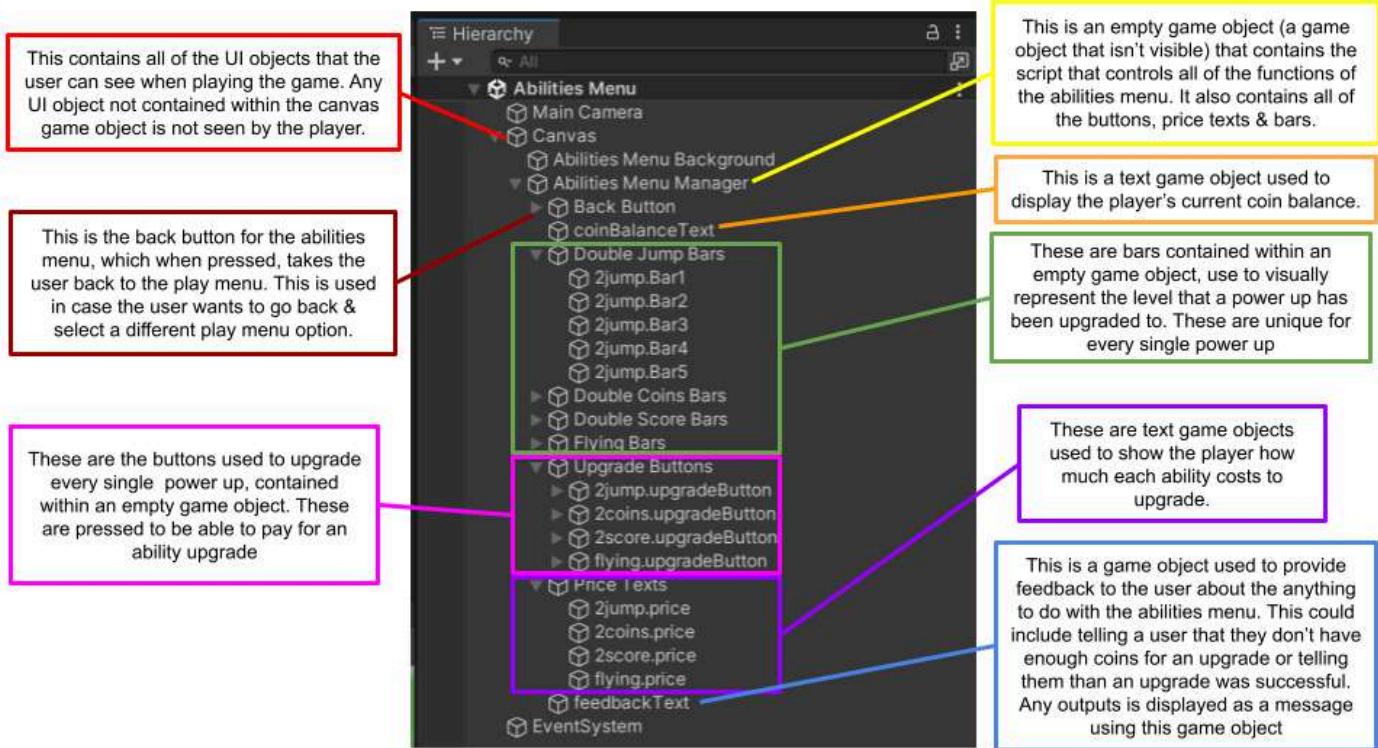
This meant that the names in the inspector matched the code for this section (shown on the right), and so when the program is searching for the name, it will search without the quotation marks.

After changing this, I tested it again, and it gave the expected outcome. Therefore, I believe I have rectified all of the errors on this module of code.

```
3 references
private void UpdatePlayerIcon(string characterName)
{
    switch (characterName)
    {
        // changes the image of the player icon image depending on the character name
        case "MaleAstronaut":
            playerIconImage.sprite = maleAstronautIcon;
            break;
        case "FemaleAstronaut":
            playerIconImage.sprite = femaleAstronautIcon;
            break;
        case "Star":
            playerIconImage.sprite = starIcon;
            break;
        case "Alien":
            playerIconImage.sprite = alienIcon;
            break;
        default:
            // debug log for any errors
            Debug.LogWarning("Unknown character selected: " + characterName);
            break;
    }
}
```

4.4 - Abilities Menu Development

4.4.1 - Abilities Menu Game Objects



4.4.2 – Abilities Menu Code

Lines 11 to 40 of Abilities Menu Code

```

11  public class AbilitiesMenuManager : MonoBehaviour
12  {
13      // references
14      public TMP_Text coinBalanceText; // text element to show coin balance
15      public Button[] upgradeButtons; // creating an array for upgrade buttons for each power-up
16      public GameObject[] doubleJumpBars, doubleCoinsBars, doubleScoreBars, flyingBars; // the actual bars for each power up
17      public TMP_Text[] priceTexts; // text elements to display prices for each power up upgrade
18      public Button backButton; // back button to go to play menu
19
20      // references
21      private int coinBalance = 0;
22      private int[] powerUpLevels = { 1, 1, 1, 1 }; // setting default power up levels to be 1
23      private int[] prices = { 50, 70, 100, 200 }; // price for each upgrade level
24      private float[] durations = { 30f, 40f, 50f, 60f, 120f }; // duration of each power up after levelling up
25      private const string coinBalanceKey = "CoinBalance"; // field name in playfab
26      private const string powerUpKey = "PowerUpLevels"; // field name in playfab
27      public TMP_Text feedbackText; // text element for feedback messages
28
29      private void Start()
30      {
31          //retrieving using info from playfab when menu is opened
32          RetrieveCoinBalance();
33          RetrievePowerUpLevels();
34          UpdateUI(); // updates menu UI based on info retrieved
35
36          // subroutine for back button
37          public void GoBack()
38          {
39              SceneManager.LoadScene("Play Menu");
40          }

```

This is the declaration of the variables, arrays and game objects to be used within the abilities menu code. The declaration of these allows them to be accessed within the code, and therefore facilitates the fulfilment of success criteria 9, 22, 23, 25 & 33.

The second section of this code is the code that runs when the abilities menu is first opened. Any code within the 'Start()' subroutine is automatically run by Unity first. This code contains 3 other subroutines 'RetrieveCoinBalance()', 'RetrievePowerUpLevels()' and 'UpdateUI()'. These subroutines are declared later, but are called when the menu is opened and all contribute to success criteria number 9, the abilities menu UI. This is because the information returned from these subroutines are used to later on display all of the information to the user on-screen, and makes up the abilities menu UI.

The final section of this code is the code that is attached to the back button. This code utilises Unity's scene management and takes the user to the play menu when pressed. This contributes to success criteria number 12, "Back buttons on all menus", as it allows the user to go back to the Play Menu and see other options.

Lines 42 to 80 of Abilities Menu Code

```

42 // retrieves the user's coin balance from PlayFab
43 [SerializeField]
44 public void RetrieveCoinBalance()
45 {
46     PlayFabClientAPI.GetUserDatatnew GetUserDataRequest(), OnUserDataSuccess, OnPlayFabError);
47 }
48
49 //subroutine for successful data retrieval
50 [SerializeField]
51 private void OnUserDataSuccess(GetUserDataResult result)
52 {
53     // if coin balance is found then set it to variable declared earlier
54     if (result.Data != null && result.Data.ContainsKey(coinBalanceKey))
55     {
56         coinBalance = int.Parse(result.Data[coinBalanceKey].Value);
57     }
58     else
59     {
60         coinBalance = 0; // set to default if no coin balance found
61         SaveCoinBalance();
62     }
63     UpdateCoinBalanceText();
64     UpdateUI();
65 }
66
67 // alerts user of any errors
68 [SerializeField]
69 private void OnPlayFabError(PlayFabError error)
70 {
71     Debug.LogError("Error retrieving data from PlayFab: " + error.GenerateErrorReport());
72 }
73
74 // saves player's coin balance to PlayFab
75 [SerializeField]
76 public void SaveCoinBalance()
77 {
78     var request = new UpdateUserDataRequest
79     {
80         Data = new Dictionary<string, string> { { coinBalanceKey, coinBalance.ToString() } }
81     };
82     PlayFabClientAPI.UpdateUserData(request, OnSaveUserDataSuccess, OnPlayFabError);
83 }

```

The first section of this code is a subroutine that retrieves the coin balance of the player from PlayFab. This links to success criteria number 22, "Coins balance retrieved from external database in abilities menu", as it creates a new PlayFab user data request and retrieves the coin balance data from the external database. If this retrieval is successful, a subroutine (declared later) called 'OnUserDataSuccess' is run, and if not, then a subroutine called 'OnPlayFabError' is run.

The second section of this code is the 'OnUserDataSuccess' subroutine being declared. This is run when the retrieval of coin balance data from PlayFab is

successful. It first checks to see if the data retrieved was from the coin balance field in PlayFab, and that it isn't null. If it is null, this means that the user does not have any coin balance data, so this means that it is set to zero, and the 'SaveCoinBalance()' subroutine is run so that it is saved to PlayFab for later use. If the field exists and is not null, the value retrieved is saved to a variable called 'coinBalance' for later use. This is so that success criteria number 23, 'Coin balance updated in the external database after every upgrade purchase', can be fulfilled, as later on this is subtracted from during purchases.

After this if statement, the subroutines 'UpdateCoinBalance()' and 'UpdateUI()' are called. These are used to display all of the new information that is retrieved from the external database, PlayFab.

The next section of the code is the subroutine that is run if there is an error with retrieving the data, a subroutine called 'OnPlayFabError'. This is mostly for maintaining the code, as it returns a message to the debug log informing the developer of the issue with the retrieval of data. Involving features that allow maintenance of the system means that in future prototypes of my game, I can easily tell if everything is running smoothly.

The last section of this code is the code that saves the player's current coin balance to PlayFab. This links to success criteria number 53, where the external database 'Stores coin balance', as it creates a new PlayFab update user data request, which is what is used to send data to the external database to store. After making this request, it calls 2 subroutines depending on the success of the request, 'OnUserDataSuccess' if the request is successful, and 'OnPlayFabError' if the request is unsuccessful.

Lines 82 to 120 of Abilities Menu Code

```

82     private void OnSaveUserDataSuccess(UpdateUserDataResult result)
83     {
84         Debug.Log("Coin balance successfully updated in PlayFab.");
85     }
86
87     // retrieves power up levels from PlayFab
88     public void RetrievePowerUpLevels()
89     {
90         PlayFabClientAPI.GetUserData(new GetUserDataRequest(), OnGetPowerUpLevelsSuccess, OnPlayFabError);
91     }
92
93     // retrieves power up levels from PlayFab
94     private void OnGetPowerUpLevelsSuccess(GetUserDataResult result)
95     {
96         if (result.Data != null && result.Data.ContainsKey(powerUpKey))
97         {
98             string[] levels = result.Data[powerUpKey].Value.Split(',');
99             for (int i = 0; i < levels.Length; i++)
100             {
101                 powerUpLevels[i] = int.Parse(levels[i]);
102             }
103         }
104         else
105         {
106             powerUpLevels = new int[] { 1, 1, 1, 1 }; // sets levels to default (1) if no data found
107             SavePowerUpLevels(); // saves the default values
108         }
109         UpdatedUI(); // updates UI after retrieving levels
110     }
111
112     // saves power up levels to PlayFab
113     public void SavePowerUpLevels()
114     {
115         string levels = string.Join(",", powerUpLevels);
116         var request = new UpdateUserDataRequest();
117         {
118             Data = new Dictionary<string, string> { { powerUpKey, levels } };
119         };
120         PlayFabClientAPI.UpdateUserData(request, OnSaveUserDataSuccess, OnPlayFabError);
121     }

```

The first section of this code is the subroutine that runs if the PlayFab user data update request is successful. This is for maintaining the system, as it has no effect on the running of the game for the user, it just sends a message to the debug log for the developer to know that the data was successfully saved.

The next section of this code is another PlayFab user data retrieval request, except this is for the respective levels of all the power ups. This links to success criteria number 59, 'Storage of power up levels for each player', as this is the application of this criteria. As the power up levels are stored, they can be retrieved using this subroutine and used later on to display information to the user. After this data is attempted to be retrieved, 1 of 2 subroutines are called. 'OnGetPowerUpLevelsSuccess' is called if the retrieval of data is successful, and 'OnPlayFabError' is called if there is an error when retrieving data.

The next section of this code is the 'OnGetPowerUpLevelsSuccess' subroutine, which takes the data retrieved from PlayFab, currently a comma-separated list, splits it where the commas are, and assigns it to an integer array 'powerUpLevels'. If there is no power up level data, then all of the levels are defaulted to be 1, and then the 'SavePowerUpLevels' subroutine is called to save this to PlayFab. The calling of this subroutine links to success criteria number 59, as this subroutine stores power up levels for each player to the external database. The last subroutine called after this is the 'UpdateUI' subroutine, which is used to display all of the retrieved data onscreen.

The last section of this code is the 'SavePowerUpLevels' subroutine. This subroutine first creates a string called 'levels' and joins all of the elements from the 'powerUpLevels' array into a comma separated string. The creation of this string is necessary, as to send the data to PlayFab, the data must be a string. A new user update request is made, and the appropriate subroutine is called based on whether the request was successful, or if there was an error. This links to success criteria number 59, 'Storage of power up levels for each player', as this is the subroutine used to carry out that specific function.

Lines 122 to 136 of
Abilities Menu Code

```

121
122     // updates the text element to display current coin balance
123     private void UpdateCoinBalanceText()
124     {
125         coinBalanceText.text = coinBalance.ToString();
126     }
127

128     // updates the bars & price texts for each power up
129     private void UpdateUI()
130     {
131         UpdatePowerUpUI(doubleJumpBars, 0, priceTexts[0]);
132         UpdatePowerUpUI(doubleCoinsBars, 1, priceTexts[1]);
133         UpdatePowerUpUI(doubleScoreBars, 2, priceTexts[2]);
134         UpdatePowerUpUI(flyingBars, 3, priceTexts[3]);
135     }
136

```

The first section of this code (as described in the comments), is the code that updates the text on the coin balance field. This takes the value within the coin balance variable, converts it to a string and then assigns it to be shown by the feedback text game object. This needs to be converted to a string because the coinBalance variable is initiated as an integer, so that arithmetic operations can be performed on it, but as the feedback text game object only takes strings, this must be converted to a string before it is displayed.

The second part of this code is the 'UpdateUI' subroutine. This is a subroutine that calls the 'UpdatePowerUpUI' 4 times, for all of the different power ups, so that the relevant information for each power up can be shown on screen. It because of the calling of the 'UpdateUI' subroutine that success criteria number 25, 'Indicator of each power up's level', can be fulfilled, as this subroutine individually calls the subroutine that updates all of the bars, which are what indicates the level of each power up.

Lines 137 to 169 of Abilities Menu Code

```

137     // updates the bars and price text for a specific power up
138     // 4 references
139     private void UpdatePowerUpUI(GameObject[] bars, int powerUpIndex, TMP_Text priceText)
140     {
141         if (bars == null || priceText == null || upgradeButtons.Length <= powerUpIndex)
142         {
143             Debug.LogWarning("Missing UI elements for power-up index: " + powerUpIndex);
144             return;
145         }
146
147         // initially hides all the bars
148         foreach (var bar in bars)
149         {
150             bar.SetActive(false);
151         }
152
153         // displays the amount bars based on the power up level
154         for (int i = 0; i < powerUpLevels[powerUpIndex]; i++)
155         {
156             bars[i].SetActive(true);
157         }
158
159         // updates the price text for the next upgrade
160         if (powerUpLevels[powerUpIndex] < prices.Length)
161         {
162             priceText.text = prices[powerUpLevels[powerUpIndex]].ToString();
163         }
164         else
165         {
166             priceText.text = "MAX"; // if max level, disable further upgrades
167             upgradeButtons[powerUpIndex].interactable = false; // disable button when max level reached
168             feedbackText.text = "You have reached the maximum power up level";
169         }
170     }

```



This section of code is used to update the user interface of the power ups. This means the fulfillment of success criteria number 25, 'Indicator of each power up's level', as it is the subroutine that manages how many bars are shown on screen, which in my game, are the representation of the power up's current level (shown on the right). This means that the existence of the bars fulfills the success criteria.

The first part of the subroutine is an if statement that checks to see that all the variables and arrays that need to be used to update the UI (prices of upgrades & the images of the buttons) are present to be utilised. If not, then the debug log is used to alert the developer that they aren't. This is beneficial, as the use of the debug log means that the system can remain maintainable for the future, as if any issues arise in any future iterations, then they will be reported in the debug log.

The next aspect of this subroutine is the initial hiding of all the bars. This ensures that only the correct bars are set to be shown according to the level of the power up. After this, in order to fully represent the power up levels and fulfill success criteria number 25, there is a for loop to show as many bars as the power up has been upgraded.

After doing these things (as written in the code comments), there is then an if statement to check if the power up has been fully upgraded. If it hasn't, then the price on the button is changed to correspond to the price of the

next upgrade. If the power up has been fully upgraded, the user is alerted through the feedback text game object, the price text changes to a string, "MAX" and the button is disabled.

Doing all of these things at the end helps to make my game more usable, as it ensures that it is clearly communicated to the user the level of their power ups, prevents them from making any accidental purchases by disabling the button, and alerts the user of this. The fact that the user is informed makes the game more usable.

Lines 170 to 191 of Abilities Menu Code

```
170
171     // subroutine to update a specific power up
172     0 references
173     public void TryUpgradePowerUp(int powerUpIndex)
174     {
175         if (powerUpLevels[powerUpIndex] < prices.Length && coinBalance >= prices[powerUpLevels[powerUpIndex]])
176         {
177             // deducts the price of the upgrade from balance
178             coinBalance -= prices[powerUpLevels[powerUpIndex]];
179
180             // updates & displays the power up level & saves the new coin balance & power up levels
181             powerUpLevels[powerUpIndex]++;
182             SaveCoinBalance();
183             SavePowerUpLevels();
184             UpdateUI();
185         }
186         else
187         {
188             feedbackText.text = "You don't have enough coins for this upgrade";
189         }
190     }
191 }
```

The last section of the abilities menu code is the subroutine that is called to upgrade a specific power up (as written in the green comments). This checks to see first if there are any more available upgrades, and then checks if the user's coin balance is greater than or equal to the price of the next upgrade. If it is both of these things, then it subtracts the price of the power up from the coin balance. This links to success criteria number 23 'Coin balance updated in external database after every upgrade purchase', as this is the variable that is used to send data back to PlayFab after a power up is purchased. The code then increases the level of the power up, and calls 3 subroutines to display these changes and save them to PlayFab.

If the user doesn't have enough coins then the feedback text game object is used to inform the user of this. I believe that the existence of feedback text makes the game more usable and accessible to users, as they know why certain things aren't possible for them, or aren't functioning as expected.

4.4.3 - Abilities Menu Iterative Testing

4.4.3.1 - Test Table

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/Failure	Remedial Actions
Initial Display of Power Up Level	34	N/A	1 light green box with '30s' on it, displayed to the player	1 light green box with '30s' on it, displayed to the player for every power up	Success	N/A
Purchase of power up	35	Click purchase button on double jump ability with no coins in balance	User should be alerted that they don't have enough coins for this purchase. Double jump level should not be increased, and display remains the same.	User is alerted that they don't have enough coins for this purchase. Double jump level isn't increased, and display remains the same.	Success	N/A
	36	Click purchase button on double jump ability with 80 coins in balance	Double jump level should increase by 1, have 2 light green boxes, show 40s on the rightmost green box, remove 50 coins from the balance and display 30.	Double jump level increased by 1, there were 2 light green boxes with it showing 40s on the rightmost box. 70 was removed from the coin balance, not 50. Displayed 80 in the coin balance even after purchase	Failure	Call the display coin balance update subroutines more often, and change the default coin price to be 50
Display of Coin Balance	37	N/A	Player's coin balance displayed on screen	Player's coin balance displayed on screen	Success	N/A

4.4.3.2 - Test Evidence



External Database

Player data	Key	Value
	PowerUpLevels	1,1,1
	CoinBalance	100
	SelectedCharacter	MaleAstronaut

Test No. 37



4.4.4 - Review of Abilities Menu Development

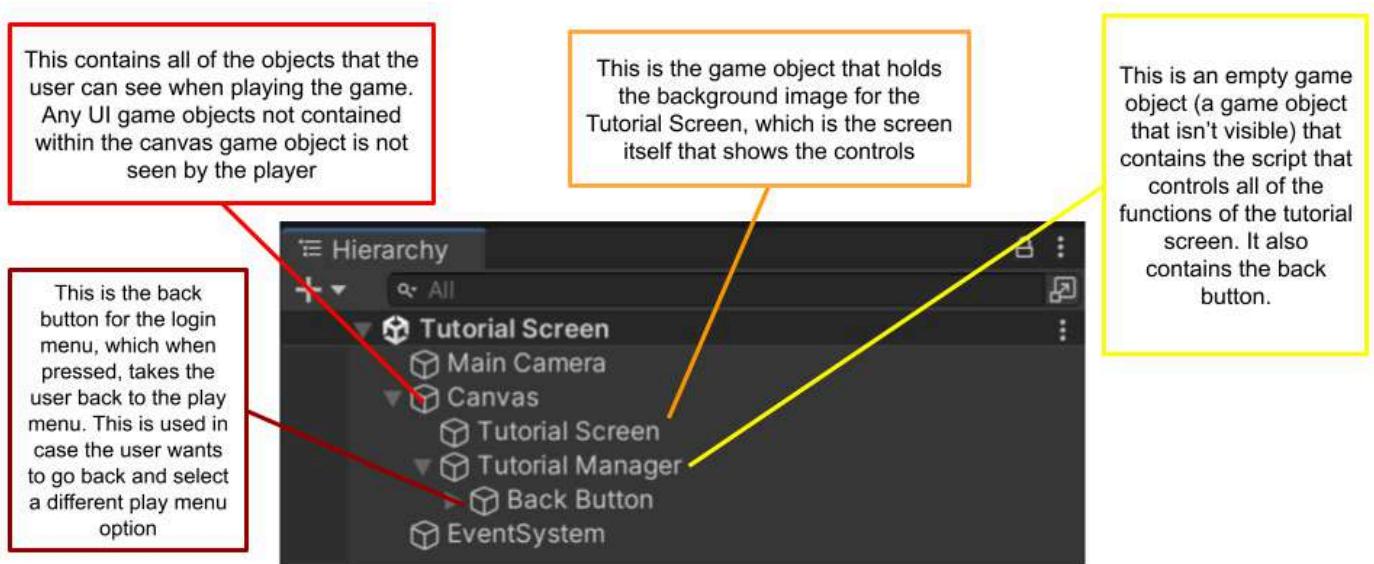
The only test that failed was test number 36, which I have decided I will rectify the issue in the final prototype of my game, AstroSprint v4.0. The remedial actions I will take were outlined in the test table, as follows: Call the display coin balance, update subroutines more often, and change the default coin price to be 50. I will call the display of coin balance subroutines more often because that will mean that after a purchase, the display is updated.

The reason why I will change the default coin price, is that displaying '70' too early offsets the duration upgrade process. It offsets it because 70 is

meant to be the 2nd upgrade price, and so the ability to upgrade will be stopped after the 4th upgrade, rather than the 5th, as described in my design. This is problematic, as it means that the user isn't able to reach the 120s duration upgrade, as the button is made unable to click after the 60s upgrade.

4.5 - Tutorial Screen Development

4.5.1 - Tutorial Screen Game Objects



4.5.2 - Tutorial Screen Code

Tutorial Screen Code

```
1 //importing libraries
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.SceneManagement;
6
7 public class TutorialManager : MonoBehaviour
8 {
9     //go back to play menu
10    public void GoBack()
11    {
12        SceneManager.LoadScene("Play Menu");
13    }
14 }
```

The first part of this code is the importing of libraries, in order utilise different functions within them, such as the Unity Scene Manager to change between scenes. The rest of this code is the 'GoBack' subroutine, which uses the Unity Scene Manager to go back to the Play Menu when the button is pressed.

This relates to success criteria number 12, 'Back buttons on all menus', as this subroutine is attached to the back button game object's 'onclick' function.

4.5.3 - Tutorial Screen Testing

4.5.3.1 - Test Table

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/ Failure	Remedial Actions
Back Button	32	Back button pressed	User is taken back to play menu	User is taken back to play menu	Success	N/A
	33	Random (non-button) area of screen pressed	Nothing happens	Nothing happens	Success	N/A

4.5.3.2 - Test Evidence



4.5.4 - Review of Tutorial Screen Development

All of my tests were successful. However, while testing I noticed that the name of the screen ('Tutorial') is slightly misleading in its nature.

This is because the definition of 'tutorial' (according to <https://languages.oup.com>) is "an account or explanation of a subject or task, especially as an online video", which is not something that my tutorial does. Acknowledging this, I have decided to make a change for my final iteration, in which I will call it 'instructions' rather than 'tutorial'. This is because (according to <https://languages.oup.com/>), the definition of 'instructions' is "detailed information about how something should be done or operated", which is much closer to the function of the Tutorial Screen.

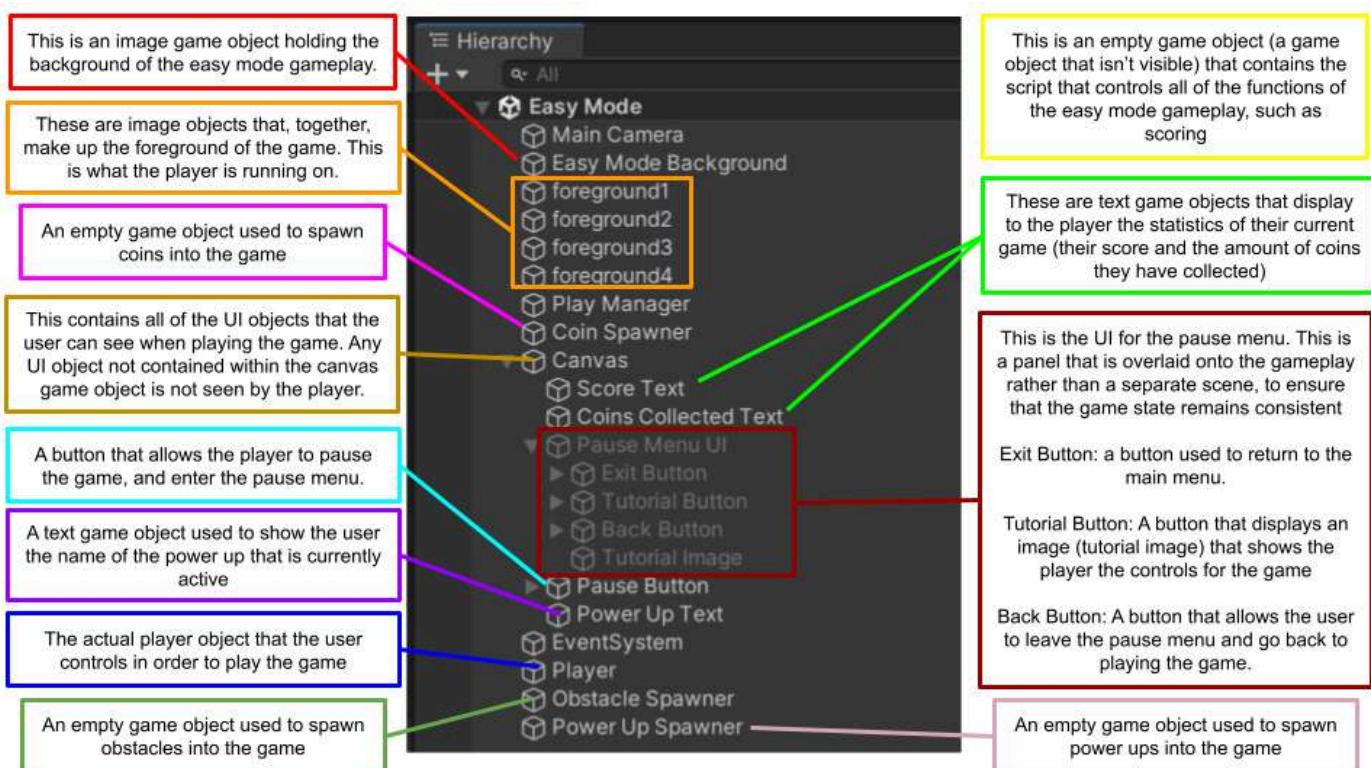
5 - Third Iteration Development (AstroSprint v3.0)

5.1 - AstroSprint v3.0 Summary

Below is the development of the third iteration of my code, which includes the previously developed menus (Main Menu, Login Menu, Sign Up Menu, Play Menu, Character Menu & Abilities Menu), as well as 2 additional menus (Pause Menu & Game Summary Menu) ad well as the actual gameplay. All together, these make up AstroSprint v3.0.

5.2 - Easy Mode Development

5.2.1 - Easy Mode Game Objects



4.6.2 - Easy Mode Code

5.2.2.1 - Coin Spawner Script

Coin Spawner Code

```
1  using UnityEngine;
2
3  public class CoinSpawner : MonoBehaviour
4  {
5      public GameObject coinPrefab; //declaring coin prefab to later instantiate
6
7      [SerializeField]
8      public float spawnInterval = 5f; //interval between coin spawns
9
10     public float spawnHeight = 10f; //height that coins spawn at
11
12     private Camera mainCamera;
13
14     void Start()
15     {
16         mainCamera = Camera.main; //get the main camera
17
18         //start spawning coins at regular intervals
19         InvokeRepeating("SpawnCoin", 0f, spawnInterval);
20     }
21
22     void SpawnCoin()
23     {
24         //get the camera's size in world units
25         float cameraWidth = mainCamera.orthographicSize * 2f * mainCamera.aspect;
26         float cameraHeight = mainCamera.orthographicSize * 2f;
27
28         //generate a random position for the coin within the camera's bounds
29         float randomX = Random.Range(-cameraWidth / 2, cameraWidth / 2); //generate random x-coordinates
30         Vector3 spawnPosition = new Vector3(randomX, spawnHeight, 0f);
31
32         //instantiate the coin at the random position
33         Instantiate(coinPrefab, spawnPosition, Quaternion.identity);
34     }
35 }
```

The code is divided into three sections, each highlighted with a colored border:

- Section 1:** Lines 1-13. This section includes the using statement, the class definition, variable declarations for the coin prefab, spawn interval, and spawn height, and the declaration of the main camera.
- Section 2:** Lines 14-20. This section contains the Start() method, which retrieves the main camera and begins spawning coins at regular intervals using the InvokeRepeating function.
- Section 3:** Lines 21-35. This section contains the SpawnCoin() method, which calculates the camera's size in world units, generates a random position within the camera's bounds, and then instantiates the coin prefab at that position.

The first section of this code is the declaration of the 'coinPrefab' game object, the variables 'spawnInterval' and 'spawnHeight', as well as the main camera. The declaration of the game object 'coinPrefab' is most essential, as it means that the coin spawner (an empty game object) can create instances of coins (which is essentially spawning coins into the game). The variable 'spawnInterval' is necessary, as it is a set value that defines the time period between the spawning of coins. This is necessary because it means that not too many or too few coins are spawned into the game. The 'spawnHeight' variable is necessary to ensure that all of the coins start from the same consistent height. The camera is also necessary to declare here, to ensure that, when spawning, the coins spawn within the bounds of the camera so the player can see them.

The next section of this code is the 'Start' subroutine, a subroutine which is run immediately by unity when the menu is opened. This gets the main camera and its dimensions, and uses the 'InvokeRepeating' function to call the 'SpawnCoin' subroutine at the specified interval (defined by the variable

'spawnInterval'). The 'Start' subroutine is necessary so that coins can begin to be spawned as soon as the player enters Easy Mode.

The last section of this code is the subroutine that actually spawns the coin. This utilises the camera (declared in the first section) and gets its height and width, and assigns it to variables 'cameraWidth' and 'cameraHeight'. These are necessary to declare because the side of the camera is used to spawn the coins in a given range inside of the camera. The next part of this code is generating the coordinates of where the coin will be spawned, which is randomly generated between the camera bounds so that the game isn't predictable in spawning all of the coins in the same position. After the coordinates are decided, they are assigned to a new variable called 'spawnPosition', which is used within the 'Instantiate' function to spawn in a new coin.

5.2.2.2 - Coin Prefab Script

Coin Prefab Code

```

using UnityEngine;
public class CoinPrefab : MonoBehaviour
{
    //Variables
    public float fallSpeed = 4f; //speed that coin falls
    private float destroyTime = 30f; //time before the coin is destroyed
    private float timer = 0f; //timer to count up to destroyTime
    private Rigidbody2D rb;
    void Awake()
    {
        rb = GetComponent();
        rb.gravityScale = 8; //Disable default gravity
        rb.velocity = Vector2.down + fallSpeed; //apply custom fall speed to rigidbody
    }
    void Update()
    {
        //Increase the timer every frame
        timer += Time.deltaTime;
        //When the timer reaches the destroy time it destroy the coin
        if (timer == destroyTime)
        {
            Destroy(gameObject);
        }
    }
    void OnCollisionEnter2D(Collision2D collision)
    {
        //Destroy the coin if it collides with the player
        if (collision.gameObject.CompareTag("Player"))
        {
            Destroy(gameObject);
        }
        //Stop the coin's movement when it hits the ground
        if (collision.gameObject.CompareTag("Ground"))
        {
            rb.velocity = Vector2.zero;
        }
    }
}

```

As this code is assigned to the Coin Prefab, every single coin that is instantiated has this code attached to it by default.

The first section of this code is the declaration of variables (and a Rigidbody, for falling physics) to be used within the coin prefab code. The variable 'fallSpeed' is necessary in determining the speed at which coins fall from their spawn point on screen. It is necessary that this is a custom value and the coins do not accelerate at the acceleration of free fall (9.81ms^{-2}) so that the user can clearly see and plan to collect the coins. It ensures that, in Easy Mode, it is easier to collect the coins. The variable 'destroyTime' (as described in the green comment) is used to set how long a

coin will be in-game before it is destroyed. This is necessary so that the system is not overloaded with too many coins being present.

The next section of this code is the 'Awake()' function, which is a function that is called the moment the object is instantiated and is not called after that. Within this function, the default gravity is disabled (necessary so coins can fall at a set, slower pace) and the velocity of the coin falling is set to be at the fall speed that is declared in the beginning of the code. This being called in the 'Awake' function ensures that the coin abides by these features from the instant it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine. This is a subroutine that is automatically called by Unity once per frame. Within this subroutine, a timer (necessary to keep track of how long the coin has been instantiated for), is added to using the 'Time.deltaTime' function. The 'Time.deltaTime' keeps track of how much time has passed since the previous frame, and by adding this value to the overall timer variable, it can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of this coin being active has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the coin is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with a coin. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as power ups or obstacles). A collision with the player will only destroy the coin prefab, as the collection associated with this is outlined in the Player script. If the collision was with the ground, then the coin stops moving. This is necessary so that the coins don't fall through the ground.

5.2.2.3 - Foreground Movement Script

Foreground Movement Code

```
1 // Importing standard libraries
2 using System;
3 using System.Collections;
4 using System.Collections.Generic;
5 using UnityEngine;
6
7 public class ForegroundMover1 : MonoBehaviour
8 {
9     [SerializeField]
10    public float speed = 4f; // speed that foreground moves
11
12    [SerializeField]
13    public float loopPositionX = 17.7905f; // The x-position to respawn the next foreground
14
15    [SerializeField]
16    public float resetPositionX = -53.38f; // The x-position where the foreground will reset
17
18    [SerializeField]
19    public float acceleration = 0.1f; // rate at which the speed increases
20
21
22    void Update()
23    {
24        // gradual increase in speed
25        speed += acceleration * Time.deltaTime;
26
27        // moves the foreground to the left
28        transform.Translate(Vector3.left * speed * Time.deltaTime);
29
30        // if the foreground moves past the reset position, loop it back
31        if (transform.position.x < resetPositionX)
32        {
33            transform.position = new Vector3(loopPositionX, transform.position.y, transform.position.z);
34        }
35    }
36 }
```

The first section of this code begins by importing libraries in order to access the different functions of Unity. The code then begins by establishing variables to be used within this code: 'speed', 'loopPositionX', 'resetPositionX' and 'acceleration'. These variables are necessary for the following reasons:

- speed: is the speed at which the foreground begins to move, it is how far across the foreground game objects are moved per second
- loopPositionX: The position at which the next foreground object is spawned at, in order to create a smooth flow between all foregrounds
- resetPositionX: The furthest position of foreground before it is reset to the loop position
- acceleration: The rate at which the speed of the foreground increases per frame

The second section of this code is the 'Update' subroutine, which is called once per frame. This subroutine begins by increasing the speed of the foreground by the previously declared acceleration per unit time that has passed between every frame (Time.deltaTime being the change in time). This is necessary so that the foreground speeds up and the game gets more challenging as the player continues, so the game doesn't get boring and there is always an element of difficulty to ensure players stay engaged. The next thing that this code does is use the 'transform.Translate' function to actually translate the foreground and move it according to the speed contained in the speed variable. The last part of this code checks whether the foreground object should be looped back to the beginning or not. When the position of the foreground object (there being 4, all with this same script attached to them), is at the reset position, its position is set to be the loop position, so that it can be moved back around and the game can remain

endless. This links to success criteria number '38', 'Game is endless', as it is the looping of the foreground that makes the game endless.

5.2.2.4 - Obstacle Spawner Script

Obstacle Spawner Code

1

```
0 references
1 public class ObstacleSpawner : MonoBehaviour
2 {
3     // References
4     public GameObject obstaclePrefab; //prefab to spawn
5     public float spawnInterval = 2f; //time interval between spawn
6     public float yPosition = -9.48f; //y position for obstacles spawning
7
8     // references
9     private Camera mainCamera;
10 }
```

2

```
11
12     void Start()
13     {
14         mainCamera = Camera.main;
15         StartCoroutine(SpawnObstacles());
16     }
17
18     // coroutine to spawn obstacles periodically by calling subroutine periodically
19     IEnumerator SpawnObstacles()
20     {
21         while (true)
22         {
23             SpawnObstacle();
24             yield return new WaitForSeconds(spawnInterval);
25         }
26     }
27 }
```

3

```
28
29     void SpawnObstacle()
30     {
31         if (obstaclePrefab == null || mainCamera == null) return;
32
33         // gets bounds of the main camera
34         float midX = mainCamera.ViewportToWorldPoint(new Vector3(0.5f, 0, 0)).x; //centre of the screen
35         float max = mainCamera.ViewportToWorldPoint(new Vector3(1, 0, 0)).x; //right side of the screen
36
37         //generate a random x-position
38         float randomX = Random.Range(midX, max);
39
40         //set the spawn position
41         Vector3 spawnPosition = new Vector3(randomX, yPosition, 0);
42
43         //instantiate the obstacle
44         Instantiate(obstaclePrefab, spawnPosition, Quaternion.identity);
45     }
46 }
```

4

```
47 }
```

The first section of this code is the declaration of the necessary variables, the obstacle prefab, and the main camera to make the obstacle spawner. All of these are necessary to be declared so that they can be utilised for their various purposes later on within the code. The obstacle prefab is the actual object itself, and this code creates instances of it within the game. The declaration of this is necessary because this is the actual object that is the obstacle. The variables in this code ('spawnInterval' and 'yPosition') are both used to manage the spawning of the obstacle, with 'spawnInterval' determining how often obstacles are spawned, and 'yPosition' determining at what height the obstacles are spawned. The camera is necessary to be declared to ensure obstacles are spawned within its bounds.

The second section of this code is the 'Start' function, which is called the moment the menu is opened. Within this function, the camera that was previously declared is stored within a variable for later use (to get its dimensions for a random spawn position). The other piece of code within this section begins the coroutine 'SpawnObstacles'. The use of a coroutine is necessary, as it allows the spawner to wait between spawns without altering the overall timing of the game.

The next section of this code is the IEnumerator function 'SpawnObstacles', which is used to control the rate at which the obstacles are spawned. It begins by calling the 'SpawnObstacle' subroutine, which is used to actually

spawn in the obstacles. With the 'IEnumerator' function, there is access to the 'WaitForSeconds' function, which is used to make the while loop within subroutine wait for a specified amount of time (the length specified in the 'spawnInterval' variable) before looping again and calling the 'SpawnObstacle' subroutine again.

The last section of this code is the subroutine that actually spawns the obstacle. This utilises the camera (declared in the first section) and gets its width, and assigns it to variables 'minX' and 'maxX' to ensure that they never spawn on top of the player or directly in front of them so the game remains fair. These also are necessary to declare because the side of the camera is used to spawn the obstacle in a given range inside of the camera. The next part of this code is generating the coordinates of where the obstacle will be spawned, which is randomly generated between the camera bounds so that the game isn't predictable in spawning all of the obstacles in the same position. After the coordinates are decided, they are assigned to a new variable called 'spawnPosition', which is used within the 'Instantiate' function to spawn in a new obstacle.

5.2.2.5 - Obstacle Prefab Script



The first section of this code is the declaration of the variables to be used within the code ('fallSpeed', 'destroyTime' and 'timer'), all of which have their necessity to the program described in the comments (green text). The declaration of the Rigidbody2D is also in this section, which is necessary in order for the obstacles to have physics and fall at certain speeds.

The next section of this code is the 'Awake' subroutine, which is run once, the moment that an obstacle is instantiated. This function includes the disabling of gravity for obstacles, and then setting them to fall at the custom fallSpeed. This is necessary as it means obstacles fall at a set, slower pace, as to not make the game too difficult. This being established in the 'Awake' function ensures that the obstacle abides by these features from the instant it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine, a subroutine that is automatically called by Unity once per frame. Within 'Update', a timer (necessary to keep track of how long the obstacle has been instantiated for), is added to using the 'Time.deltaTime' function. This function is used to keep track of how much time has passed since the previous frame, and by adding this value to the overall timer variable, it can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of the obstacle existing in-game and if it has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the obstacle is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with an obstacle. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as power ups or coins). A collision with the player will only destroy the obstacle prefab, as the player death associated with this is outlined in the Player script. If the collision was with the ground, then the obstacle stops moving. This is necessary so that the obstacles don't fall through the ground.

5.2.2.6 - Play Manager Script

Lines 5 to 21 of Play Manager Code

```
5  public class PlayerController : MonoBehaviour
6  {
7      2 references
8      public TMP_Text scoreText;
9      3 references
10     public float currentScore = 0f;
11     2 references
12     public float baseScore = 10f;
13     2 references
14     public float timer = 0f;
15     2 references
16     public float addScoreInterval = 1f; // time interval in seconds to add score
17     3 references
18     private float nextScoreTime = 0f; // time when score should next increase
19
20
21
22
```

1

```
15     // declared to able to interact with player script
16     [SerializeField] private PlayerPrefab playerPrefabScript;
17
18     void Start()
19     {
20         nextScoreTime = addScoreInterval;
21     }
22
```

2

The first section of this code is the declaration of variables to be used in the Play Manager script. This begins with the declaration of the 'scoreText' game object, which is necessary because it is the object that allows the score to be displayed on-screen. There is then the declaration of:

- currentScore: Necessary as it holds the value of the current score of the player
- baseScore: Necessary as it holds the value that is added to the current score every second
- timer: Necessary, as it times how long the game has been running
- addScoreInterval: Necessary as it holds the value of the time interval between the adding of the base score
- nextScoreTime: Necessary as it holds the value of the next time (in seconds) that the base score will be added to the current score

All of these variable declarations also link to the fulfillment of success criteria number 34, scoring based on how long the player lasts in-game.

The next section of this code is the declaration of the player script, so that the play manager can access it. The other part of this code is the 'Start' subroutine, which is run by Unity the moment the gameplay is opened. In this subroutine, the nextScoreTime variable is set to be the value of addScoreInterval. This is necessary as by doing this, the next time that score is added is after 1 second, and not adding score immediately when the gameplay starts, as this would offset the entire scoring process.

Lines 23 to 52 of Play Manager Code

```
23 void Update()
24 {
25     timer += Time.deltaTime;
26
27     // Add score at regular intervals
28     if (timer >= nextScoreTime)
29     {
30         if (playerPrefabScript != null && playerPrefabScript.doubleScoreActive)
31         {
32             currentScore += baseScore * 2; // Double the score added if the double score is true
33         }
34         else
35         {
36             currentScore += baseScore; // regular score addition
37         }
38
39         UpdateScoreUI();
40         nextScoreTime += addScoreInterval; // schedules the next score increment
41     }
42 }
43
44 // shows score onscreen
45 public void UpdateScoreUI()
46 {
47     if (scoreText != null)
48     {
49         scoreText.text = "Score: " + Mathf.RoundToInt(currentScore);
50     }
51 }
52 }
```

1

2

The first section of this code is the 'Update' subroutine, which is run by Unity every time there is a new frame. At the beginning of this subroutine, the timer is added to by the 'Time.deltaTime' function, which is a function that gets the amount of time since the last frame was shown. This is necessary, as the repeated calling of this function means that the timer holds the overall time (in seconds) that has passed during the gameplay. The next part of this section is checking if the timer is at the same time as the nextScoreTime. This is essentially checking if it is time to add the base score again. It first checks if the double score power up is active, and if it is, then the two of the baseScore are added to the current score ($baseScore * 2$). If the double score power up is not active, then a single baseScore is added to the currentScore. After this, the 'UpdateScoreUI' subroutine is called to update the score text shown on-screen. Then the nextScoreTime has addScoreInterval added to it, so that it holds the next time that score will be added to the current score. This is necessary so that the nextScoreTime is constantly updated to represent the next point in time, which is continually passing as time progresses.

The second section of this code is the 'UpdateScoreUI' subroutine. This subroutine sets the value of the value of the 'scoreText' game object to the current score, prefixed with "Score: " so it is clear to the user what it represents. This subroutine is necessary as it is the subroutine that updates the scoreText game object on the screen so that the player can see their current score.

5.2.2.7 - Power Up Spawner Script

Power Up Spawner Code

```

1 //resources
2 public class PowerUpSpawner : MonoBehaviour
3 {
4     //variables
5     public GameObject[] powerupPrefabs; //array of power up prefabs
6     public float spawnInterval = 7f; //interval between spawns
7     public float yPosition = 5f; //y position for spawning power ups
8     private Camera mainCamera;
9 }
10
11 void Start()
12 {
13     mainCamera = Camera.main;
14     StartCoroutine(SpawnPowerUps());
15 }
16
17 //coroutine to spawn power ups periodically
18 IEnumerator SpawnPowerUps()
19 {
20     while (true)
21     {
22         SpawnPowerUp();
23         yield return new WaitForSeconds(spawnInterval);
24     }
25 }
26
27 void SpawnPowerUp()
28 {
29     if (powerupPrefabs.Length == 0 || mainCamera == null) return;
30
31     //get the bounds of the main camera
32     Vector3 min = mainCamera.ViewportToWorldPoint(new Vector3(0, 0, 0).xz); //center of the screen
33     float maxZ = mainCamera.ViewportToWorldPoint(new Vector3(1, 0, 0).xz); //right side of the screen
34
35     //generate random x-position
36     float randomX = Random.Range(minX, maxX);
37
38     //set spawn position
39     Vector3 spawnPosition = new Vector3(randomX, yPosition, 0);
40
41     //chooses a random prefab from the array
42     int randomIndex = Random.Range(0, powerupPrefabs.Length);
43     GameObject selectedPowerUp = powerupPrefabs[randomIndex];
44
45     //instantiates the powerup
46     Instantiate(selectedPowerUp, spawnPosition, Quaternion.identity);
47 }
48

```

The first section of this code is the declaration of the array of game objects 'powerupPrefab' game object, the variables 'spawnInterval' and 'yPosition', as well as the main camera. The declaration of the array of game objects is most essential, as it means that the power up spawner (an empty game object) can pick from the array to create instances of power ups (which is essentially spawning the selected power up into the game). The variable 'spawnInterval' is necessary, as it is a set value that defines the time period between the spawning of power ups. This is necessary because it means that not too many or too few power ups are spawned into the game at one time. The 'yPosition' variable is necessary to ensure that all of the power ups start from the same consistent height. The camera is also necessary to declare here, to ensure that, when spawning, the power ups spawn within the bounds of the camera so the player can see them.

The second section of this code is the 'Start' function, which is called the moment the menu is opened. Within this function, the camera that was previously declared is stored within a variable for later use (to get its dimensions for a random spawn position). The other piece of code within this section begins the coroutine 'SpawnPowerUps'. The use of a coroutine is necessary, as it allows the spawner to wait between spawning of a random power up without altering the overall timing of the game.

The next section of this code is the IEnumarator function 'SpawnPowerUps', which is used to control the rate at which the power ups are spawned. It begins by calling the 'SpawnPowerUp' subroutine, which is used to actually spawn in the power ups. With the 'IEnumarator' function, there is access to the 'WaitForSeconds' function, which is used to make the while loop within

subroutine wait for a specified amount of time (the length specified in the 'spawnInterval' variable) before looping again and calling the 'SpawnPowerUp' subroutine to instantiate a random power up again.

The last section of this code is the subroutine that actually spawns the power up. This utilises the camera (declared in the first section) and gets its width, and assigns it to variables 'minX' and 'maxX' to ensure that they never spawn on top of or behind the player so the game remains fair. These also are necessary to declare because the side of the camera is used to spawn the power up in a given range inside of the camera. The next part of this code is generating the coordinates of where the power up will be spawned, which is randomly generated between the camera bounds so that the game isn't predictable in spawning all of the power ups in the same position. After the coordinates and which power up to spawn have been determined, they are assigned to a new variable called 'spawnPosition', which is used within the 'Instantiate' function to spawn in a new power up.

The power up to be instantiated is selected through the random generation of an integer, which is taken as the index of the power up to be spawned. This random integer, stored in the 'randomIndex' variable, is used to access the element at that index in the powerupPrefab array. This selected power up is stored in 'selectedPowerUp' and used in the instantiate function to spawn the randomly selected power up in-game. The randomisation of what power up is instantiated is necessary so that the game remains to be engaging and not be predictable. This links to success criteria number 49, 'Random generation of power ups', as it is this empty game object (the power up spawner) utilising the process of selecting a random index that allows the generation of power ups to be random.

5.2.2.8 – Double Jump Script

Double Jump Code

```

1
2
3
4

```

```

1 // References
2 public class DoubleJump : MonoBehaviour
3 {
4     // References
5     public float fallSpeed = 4f; // speed the power up falls
6     // References
7     private float destroyTime = 7f; // time before the power up is destroyed
8     // References
9     private float timer = 0f; // timer to count up to destroyTime
10    // References
11    private Rigidbody2D rb;
12
13
14    void Awake()
15    {
16        rb = GetComponent<Rigidbody2D>();
17        rb.gravityScale = 0; // disables default gravity
18        rb.velocity = Vector2.down * fallSpeed; // applies new fall speed
19    }
20
21
22    void Update()
23    {
24        timer += Time.deltaTime;
25
26        if (timer >= destroyTime)
27        {
28            Destroy(gameObject); // destroys the power up after the timer reaches destroyTime
29        }
30    }
31
32    void OnCollisionEnter2D(Collision2D collision)
33    {
34        if (collision.gameObject.CompareTag("Player"))
35        {
36            Destroy(gameObject); // destroy the power up on player collection
37        }
38    }
39

```

The first section of this code is the declaration of variables (and a Rigidbody, for falling physics) to be used within the double jump prefab code. The variable 'fallSpeed' is necessary in determining the speed at which the icon for this power up falls from their respective spawn point. It is necessary that this is a custom value and the icon does not accelerate at the acceleration of free fall (9.81ms^{-2}), rather at a slower value, so that the user can clearly see and plan to collect this power up. It ensures that, in Easy Mode, it is easier to collect the power up. The variable 'destroyTime' (as described in the green comment) is used to set how long an icon will be in-game before it is destroyed. This is necessary so that the system is not overloaded with too many power ups being present.

The next section of this code is the 'Awake()' function, which is a function that is called the moment the object is instantiated and is not called after that. Within this function, the default gravity is disabled (necessary so the power up's icon can fall at a set, slower pace) and the velocity of the icon falling is set to be at the fall speed that is declared in the beginning of the code. This being called in the 'Awake' function ensures that the icon abides by these features from the moment it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine. This is a subroutine that is automatically called by Unity once per frame. Within this subroutine, a timer (necessary to keep track of how long the double jump icon has been instantiated for), is added to using the 'Time.deltaTime' function. The 'Time.deltaTime' keeps track of how much time has passed since the previous frame, and by adding this value to the overall timer variable, it

can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of this icon being active has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the double jump icon is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with the double jump icon. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as coins or obstacles). A collision with the player will only destroy the double jump prefab, as the activation of the power up itself is outlined in the Player script.

5.2.2.9 - Double Coins Script

Double Coins Code

```

1 references
2 public class DoubleCoins : MonoBehaviour
3 {
4     //reference
5     public float fallSpeed = 4f; //the speed that the power up falls
6     //reference
7     private float destroyTime = 7f; //time before the power up is destroyed
8     //references
9     private float timer = 0f; //timer to count up to destroyTime
10    //references
11    private Rigidbody2D rb;
12
13    void Awake()
14    {
15        rb = GetComponent<Rigidbody2D>();
16        rb.gravityScale = 0; //changes gravity to 0
17        rb.velocity = Vector2.down * fallSpeed; //set custom gravity
18    }
19
20    //uses timer to destroy power up after destroy time
21    void Update()
22    {
23        timer += Time.deltaTime;
24
25        if (timer >= destroyTime)
26        {
27            Destroy(gameObject);
28        }
29    }
30
31    //destroy power up after colliding with player
32    void OnCollisionEnter2D(Collision2D collision)
33    {
34        if (collision.gameObject.CompareTag("Player"))
35        {
36            Destroy(gameObject);
37        }
38    }
39
40 }

```

The first section of this code is the declaration of variables (and a Rigidbody, for falling physics) to be used within the double coins prefab code. The variable 'fallSpeed' is necessary in determining the speed at which the icon for this power up falls from their respective spawn point. It is necessary that this is a custom value and the icon does not accelerate at the acceleration of free fall (9.81ms^{-2}), rather at a slower value, so that the user can clearly see and plan to collect this power up. It ensures that, in Easy Mode, it is easier to collect the power up. The variable 'destroyTime' (as described in the green comment) is used to set how long an icon will be

in-game before it is destroyed. This is necessary so that the system is not overloaded with too many power ups being present.

The next section of this code is the 'Awake()' function, which is a function that is called the moment the object is instantiated and is not called after that. Within this function, the default gravity is disabled (necessary so the power up's icon can fall at a set, slower pace) and the velocity of the icon falling is set to be at the fall speed that is declared in the beginning of the code. This being called in the 'Awake' function ensures that the icon abides by these features from the moment it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine. This is a subroutine that is automatically called by Unity once per frame. Within this subroutine, a timer (necessary to keep track of how long the double coins icon has been instantiated for), is added to using the 'Time.deltaTime' function. The 'Time.deltaTime' keeps track of how much time has passed since the previous frame, and by adding this value to the overall timer variable, it can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of this icon being active has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the double coins icon is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with the double coins icon. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as coins or obstacles). A collision with the player will only destroy the double coins prefab, as the activation of the power up itself is outlined in the Player script.

5.2.2.10 - Double Score Script

Double Score Code

```

6  public class DoubleScore : MonoBehaviour
7  {
8      public float fallSpeed = 4f; //the speed that the power up falls
9      private float destroyTime = 7f; //time before power up is destroyed
10     private float timer = 0f; //timer to count up to destroytime 1
11
12     private Rigidbody2D rb;
13
14     void Awake()
15     {
16         rb = GetComponent<Rigidbody2D>();
17         rb.gravityScale = 0;
18         rb.velocity = Vector2.down * fallSpeed;
19     }
20
21     //uses timer to destroy power up after destroy time
22     void Update()
23     {
24         timer += Time.deltaTime;
25
26         if (timer >= destroyTime)
27         {
28             Destroy(gameObject);
29         }
30
31     //destroy power up after colliding with player
32     void OnCollisionEnter2D(Collision2D collision)
33     {
34
35         if (collision.gameObject.CompareTag("Player"))
36         {
37             Destroy(gameObject);
38         }
39     }

```

The first section of this code is the declaration of variables (and a Rigidbody, for falling physics) to be used within the double score prefab code. The variable 'fallSpeed' is necessary in determining the speed at which the icon for this power up falls from their respective spawn point. It is necessary that this is a custom value and the icon does not accelerate at the acceleration of free fall (9.81ms^{-2}), rather at a slower value, so that the user can clearly see and plan to collect this power up. It ensures that, in Easy Mode, it is easier to collect the power up. The variable 'destroyTime' (as described in the green comment) is used to set how long an icon will be in-game before it is destroyed. This is necessary so that the system is not overloaded with too many power ups being present.

The next section of this code is the 'Awake()' function, which is a function that is called the moment the object is instantiated and is not called after that. Within this function, the default gravity is disabled (necessary so the power up's icon can fall at a set, slower pace) and the velocity of the icon falling is set to be at the fall speed that is declared in the beginning of the code. This being called in the 'Awake' function ensures that the icon abides by these features from the moment it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine. This is a subroutine that is automatically called by Unity once per frame. Within this subroutine, a timer (necessary to keep track of how long the double score icon has been instantiated for), is added to using the 'Time.deltaTime' function. The 'Time.deltaTime' keeps track of how much time has passed since

the previous frame, and by adding this value to the overall timer variable, it can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of this icon being active has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the double score icon is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with the double score icon. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as coins or obstacles). A collision with the player will only destroy the double score prefab, as the activation of the power up itself is outlined in the Player script.

5.2.2.11 - Flying Script

Flying Code

```

1 0 references
2 public class Flying : MonoBehaviour
3 {
4     1 reference
5     public float fallSpeed = 4f; //the speed that the power up falls
6     1 reference
7     private float destroyTime = 7f; //time before power up is destroyed
8     3 references
9     private float timer = 0f; //timer to count up to destroyTime
10    2 references
11     private Rigidbody2D rb;
12
13     0 references
14     void Awake()
15     {
16         rb = GetComponent<Rigidbody2D>();
17         rb.gravityScale = 0;
18         rb.velocity = Vector2.down * fallSpeed;
19     }
20
21     0 references
22     void Update()
23     {
24         timer += Time.deltaTime;
25
26         if (timer >= destroyTime)
27         {
28             Destroy(gameObject);
29         }
30     }
31
32     0 references
33     void OnCollisionEnter2D(Collision2D collision)
34     {
35         if (collision.gameObject.CompareTag("Player"))
36         {
37             Destroy(gameObject);
38         }
39     }
40

```

The first section of this code is the declaration of variables (and a Rigidbody, for falling physics) to be used within the flying prefab code. The variable 'fallSpeed' is necessary in determining the speed at which the icon for this power up falls from their respective spawn point. It is necessary that this is a custom value and the icon does not accelerate at the acceleration of free fall (9.81ms^{-2}), rather at a slower value, so that the user can clearly see and plan to collect this power up. It ensures that, in Easy Mode, it is easier to collect the power up. The variable 'destroyTime'

(as described in the green comment) is used to set how long an icon will be in-game before it is destroyed. This is necessary so that the system is not overloaded with too many power ups being present.

The next section of this code is the 'Awake()' function, which is a function that is called the moment the object is instantiated and is not called after that. Within this function, the default gravity is disabled (necessary so the power up's icon can fall at a set, slower pace) and the velocity of the icon falling is set to be at the fall speed that is declared in the beginning of the code. This being called in the 'Awake' function ensures that the icon abides by these features from the moment it is instantiated until it is destroyed.

The next section of this code is the 'Update' subroutine. This is a subroutine that is automatically called by Unity once per frame. Within this subroutine, a timer (necessary to keep track of how long the flying icon has been instantiated for), is added to using the 'Time.deltaTime' function. The 'Time.deltaTime' keeps track of how much time has passed since the previous frame, and by adding this value to the overall timer variable, it can be used to keep track of time in-game. After adding to the timer every frame, this subroutine then checks to see if the time elapsed of this icon being active has been long enough for it to be destroyed. This is done by checking if the time on the timer is longer than or equal to 'destroyTime', in which if it is, the flying icon is destroyed.

The last section of this code is a subroutine, 'OnCollisionEnter2D', that deals with what happens if the player collides with the flying icon. This code first checks to see the tag of the object that has collided with it, to see if it is the player. This is because the coins should only be able to interact with the player, and should be unaffected by the collisions with other objects (such as coins or obstacles). A collision with the player will only destroy the flying prefab, as the activation of the power up itself is outlined in the Player script.

5.2.2.12 - Game Pause Script

Lines 7 to 39 of Pause Code

```

1 //game objects and buttons for the menu
2
3     public GameObject pauseMenuUI;
4     public GameObject tutorialImg;
5     public Button resumeButton;
6     public Button exitToPlayMenuButton;
7     public Button tutorialButton;
8     public Button pauseButton;
9
10
11
12
13
14
15
16
17     void Start()
18     {
19         // Set the pause menu panel to inactive at start
20         pauseMenuUI.SetActive(false);
21     }
22
23
24     public void OnPauseButtonClick()
25     {
26         if (pauseMenuUI.activeSelf)
27         {
28             Resume(); // resume the game
29         }
30         else
31         {
32             Pause(); // pause the game
33         }
34     }
35
36     public void Pause()
37     {
38         pauseMenuUI.SetActive(true);
39         Time.timeScale = 0f; //pause gameplay by freezing time
40     }

```

The first section of this code is the declaration of the buttons and game objects to be used within the pause menu. The declaration of each are necessary for their own reasons:

- Game Objects:
 - pauseMenuUI: Necessary to be declared so that it can be activated and deactivated when needed
 - tutorialImg: Necessary to be accessed in this script so that it can be shown or hidden depending on the pressing of a button
- Buttons:
 - resumeButton: Necessary so it can be used to set the pauseMenuUI to be inactive and resume the regular function of the game
 - exitToPlayMenuButton: Necessary so that the user is able to quit a game part way through without having to die in-game
 - tutorialButton: Necessary so that the user can see the tutorial in the pause menu
 - pauseButton: Necessary so the user can pause the game, links to success criteria number 42, as this is what allows the user to pause the game

The second section of this code is the 'Start' subroutine, which is run by Unity the moment the scene is opened. In this function, the pause menu UI is set to not be active, so that it doesn't show before the user is playing the game. This is necessary so that users can play the game and not see the pause menu (which, when active, will overlay on top of the gameplay), until they press the pause button.

The third section of this code is the code that links to the pause button. This checks whether the pause menu is active or not, in order to choose what subroutine is called. This is necessary, as the 2 subroutines ('Resume' and 'Pause') have opposite functions, one to activate the pause menu and the other to disable it. If the pause menu is active, then 'Resume' is called, and if not, then 'Pause' is called.

The next section of code is the 'Pause' subroutine. The existence of this subroutine fulfills success criteria number 42, as this is the subroutine that allows the game to be paused. This is because it sets the Pause Menu screen itself to be active, and then sets the time scale to be 0. This means that all functions based on time are stopped, which includes the movement of the foreground, the spawning of coins, power ups and obstacles, power up remaining durations, and the player's living state. As it stops all of these things while paused, this subroutine also fulfills success criteria numbers 43, 44, 45 & 46.

Lines 41 to 58 of Pause Code

```

1 reference
41     public void Resume()
42     {
43         pauseMenuUI.SetActive(false);
44         Time.timeScale = 1f; // resume the game by unfreezing time
45     }
46

0 references
47     public void ShowTutorial()
48     {
49         tutorialImg.SetActive(true);
50     }
51

0 references
52     public void ExitToPlayMenu()
53     {
54         //loads the play menu scene
55         SceneManager.LoadScene("Play Menu");
56     }
57 }
58

```

1
2
3

The last section of this code is the code that allows the user to exit the current game and go back to the play menu. If a user The first section of this code is the code that allows the user to resume the game. When the pause button is clicked, if the pause menu is active, then this subroutine is called. In this subroutine, the pause menu UI is set to be inactive again. This is necessary as the pause menu is an overlay on the actual gameplay, so needs to be set inactive to be able to see the gameplay again. The time scale is also set back to 1, so that time progresses at the rate it usually does. This means that all functions (scoring, spawning, power up durations) are back to happening at the normal rate, and not unfrozen.

The next section of this code is the 'ShowTutorial' subroutine. This is attached to the on click function of the 'tutorialButton', in order to set active the image of the tutorial, which is a smaller version of the one in the actual tutorial screen. This is necessary to make the game more usable, as it means that they can access the controls of the game anytime.

The last section of this code is the code that is attached to the 'Exit To Play Menu' button. This allows smooth navigation throughout the game, as this is an easy, efficient method of getting back to the Play Menu without the user having to force themselves to die in-game. This links to success criteria number 12, back buttons on all menus, as this is the back button for the gameplay to return to the Play Menu, even if it is not explicitly called a back button.

5.2.3 - Easy Mode Iterative Testing

5.2.3.1 - Test Table

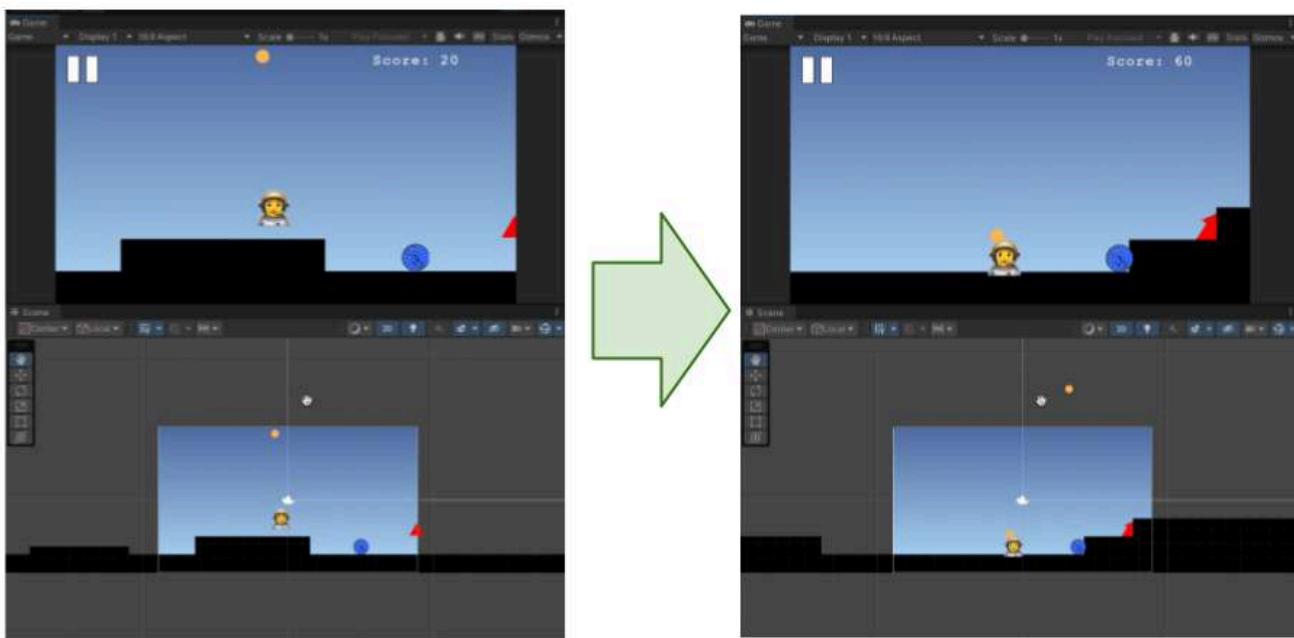
Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/Failure	Remedial Actions
Foreground Movement	39	N/A	Movement of foreground from right to left.	Foreground moved from right to left	Success	N/A
Player Movement	40	'Space' key pressed	Player jumps	Player jumped	Success	N/A
	41	Non-bind ed key pressed: 'L'	Nothing happens.	Nothing happened	Success	N/A
Obstacle Generation	42	N/A	After 5 seconds have passed in-game, an obstacle appears	After 5 seconds have passed in-game, an obstacle appears	Success	N/A
	43	N/A	Obstacles spawning at higher y-values than they would for regular movement when flying	Obstacles spawn at the same height constantly	Failure	Add code to check if the 'flying' power up is currently active, and if it is, then change the obstacle spawn height to a higher value
	44	Obstacle	Obstacles out	Obstacles out	Success	N/A

		out of frame	of frame are deleted from the Hierarchy menu in Unity	of frame were deleted from the Hierarchy menu in Unity		
Double Score Power Up	45	Player collision with double score power up icon	Score should increase by 200 in 10 seconds.	Score increased from 10 to 210 in 10 seconds	Success	N/A
	46	Waiting 30 seconds	After waiting 30 seconds for power up to wear off, the score should increase by 100 in 10 seconds.	After waiting 30 seconds for power up to wear off, the score increased by 100 in 10 seconds.	Success	N/A
Flying Power Up	47	Space key held down	The player should move up as long as the button is being held	Player moved up for as long as the space button was held	Success	N/A
	48	Waiting 30 seconds	Player falls to the floor	Player falls to the floor	Success	N/A
	49	Player collision with power up icon	Player is able to fly	Text appeared on-screen to show that the power up was active and the player was able to fly	Success	N/A
	50	Space key held pressed when player is at top of screen	Player is stopped from moving out of frame.	The player dies when they vertically move out of frame when flying	Failure	Add a collider to the top of the screen to ensure that's active when flying to ensure that the player does not go above this
	51	Player collision with coin	Player collects coin and gains 2 instead of 1	Player collected coin and gains 2	Success	N/A

		when power up is active		instead of 1		
Double Jump Power Up	52	'Space' key pressed twice without touching the floor	Player jumps twice.	Player jumped twice.	Success	N/A
Player death	61	Collision with obstacle	Player dies.	Player dies.	Success	N/A
Pausing the Game	62	Pressing pause button	The player is shown the pause menu and all of its options	The gameplay is paused, however, the pause menu is not shown	Failure	Rearrange the code inside of the pause button to make sure that the panel for the pause menu is set to active at the same time that time is paused in-game
	63	Pressing unpause button	Player returns to the gameplay	Player returned to gameplay	Success	N/A
	64	Pressing area of screen not linked to a button	Nothing	Nothing	Success	N/A
	65	Pressing the tutorial button	Tutorial image is shown on-screen	Tutorial image was shown on-screen	Success	N/A
	66	Pressing the pause button, then pressing back	Game statistics are maintained	Game statistics were maintained	Success	N/A

5.2.3.2 - Test Evidence

Test No. 39



Test No. 40



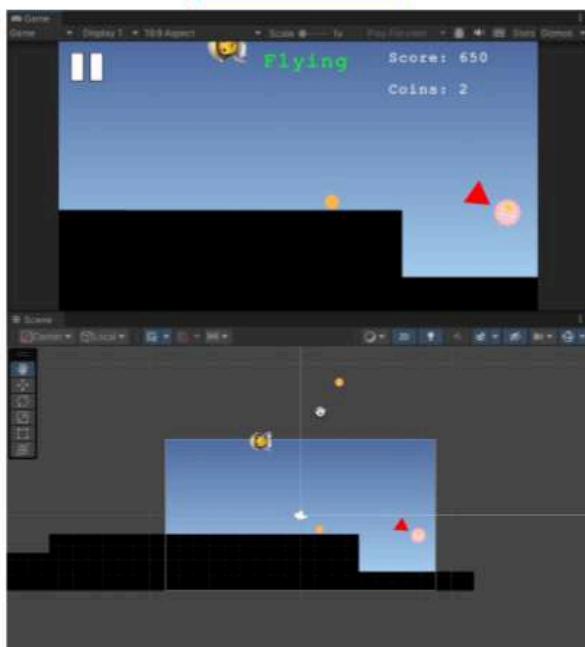
Test No. 41



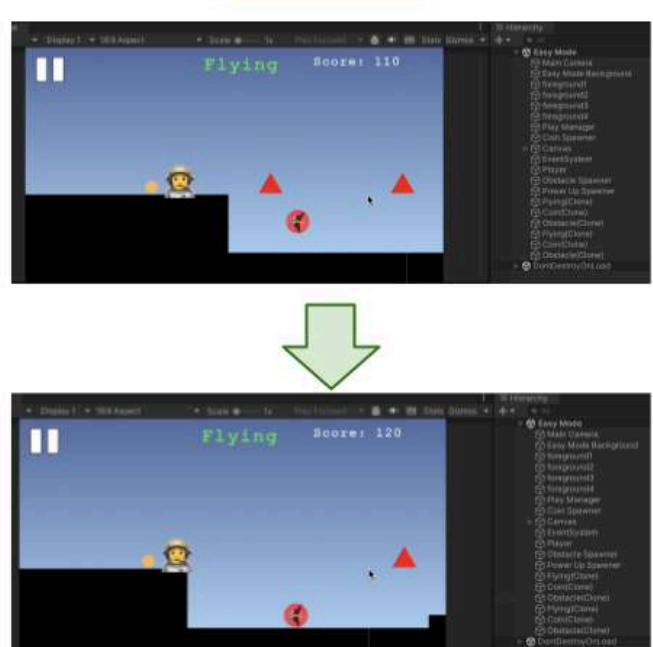
Test No. 42



Test No. 43



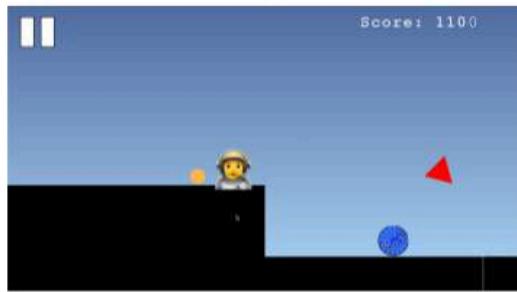
Test No. 44



Test No. 45



Test No. 46



Test No. 47



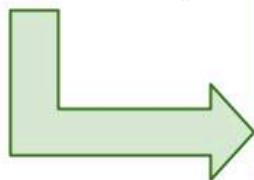
Test No. 48



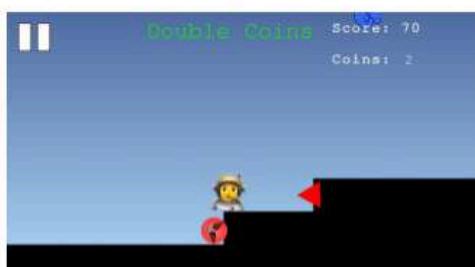
Test No. 49



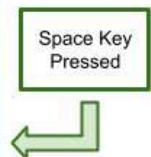
Test No. 50



Test No. 51



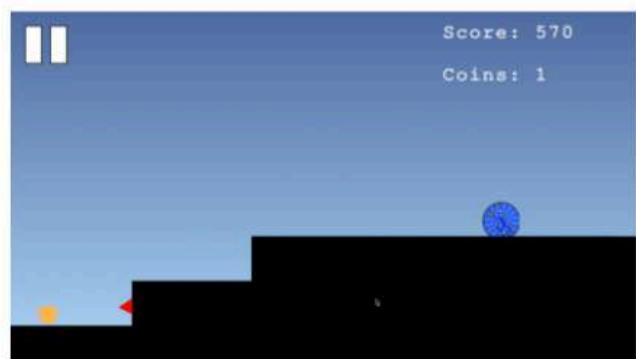
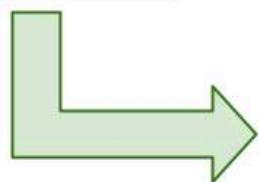
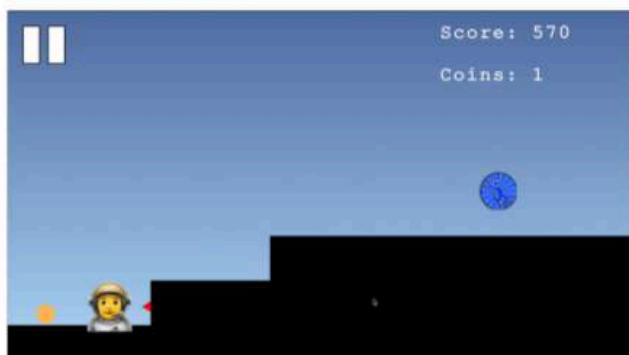
Test No. 52



Space Key Pressed



Test No. 61



Test No. 62



Test No. 63



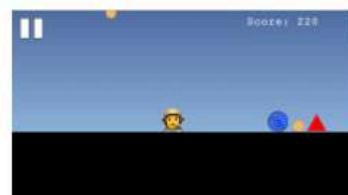
Test No. 64



Test No. 65



Test No. 66



5.2.4 - Review of Easy Mode Development

The tests that failed in the initial Development of Easy Mode are as follows:

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Remedial Actions
Obstacle Generation	43	N/A	Obstacles spawning at higher y-values than they would for regular movement when flying	Obstacles spawn at the same height constantly	Add code to check if the 'flying' power up is currently active, and if it is, then change the obstacle spawn height to a higher value
Flying Power up	50	Space key held pressed when player is at top of screen	Player is stopped from moving out of frame.	The player dies when they vertically move out of frame when flying	Add a collider to the top of the screen to ensure that's active when flying to ensure that the player does not go above this
Pausing the Game	62	Pressing pause button	The player is shown the pause menu and all of its options	The gameplay is paused, however, the pause menu is not shown	Rearrange the code inside of the pause button to make sure that the panel for the pause menu is set to active at the same time

					that time is paused in-game
--	--	--	--	--	--------------------------------

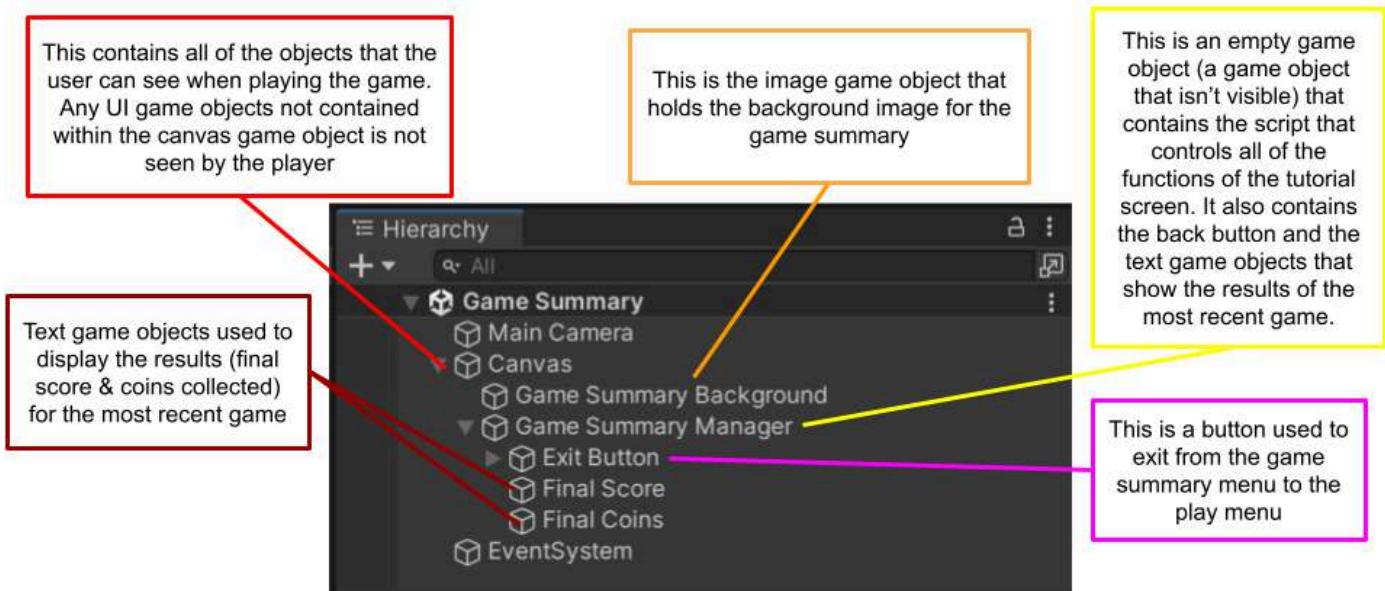
I have decided the remedial actions to be as follows:

- Test 43: Add code to check if the 'flying' power up is currently active, and if it is, then change the obstacle spawn height to a higher value
 - This is necessary, as it means that the game will remain difficult even when flying is active.
 - If the obstacles do not spawn at a higher height when flying is active than they would regularly, the player could just fly above all of the obstacles, and the game would become too easy while this power up is active.
 - Changing this feature would mean that the game remains to be challenging, and therefore engaging, even if the flying power up is active
- Test 50: Add a collider to the top of the screen to ensure that's active when flying to ensure that the player does not go above this
 - This is necessary, as a collider would prevent the player from moving out of the screen, causing them to die
 - However, I have also considered the possibility of exploring if there is a function inside of Unity that prevents the player from involuntarily exiting the camera view, but I will look into the Unity documentation to see if this is possible.
- Test 62: Rearrange the code inside of the pause button to make sure that the panel for the pause menu is set to active at the same time that time is paused in-game
 - This is essential so that the player can actually see the pause menu and utilise its functions, rather than just only freezing the game
 - If this is not fixed, it will be extremely difficult for users to understand how to access the pause menu, as they might not know that it requires 2 clicks, and this is not mentioned anywhere in the game

I have decided to implement the remedial actions needed in the final iteration of my game, AstroSprint v4.0.

5.3 - Game Summary Development

5.3.1 - Game Summary Game Objects



5.3.2 – Game Summary Code

Lines 10 to 47 of Game Summary Code

```

10 public class GameSummaryManager : MonoBehaviour
11 {
12     //exit game summary to go to play menu
13     0 references
14     public void ExitToPlayMenu()
15     {
16         SceneManager.LoadScene("Play Menu");
17     }
18     //text elements to show final score & final coins on screen
19     1 reference
20     public TMP_Text finalCoinsText;
21     1 reference
22     public TMP_Text finalScoreText;
23     2 references
24     private int finalCoins;
25     2 references
26     public float finalScore;
27     0 references
28     void Start()
29     {
30         FetchCurrentCoinsCollected();
31         FetchCurrentScore();
32     }
33     //FETCHES "CurrentCoinsCollected" from PlayFab and saves it to finalCoins
34     1 reference
35     void FetchCurrentCoinsCollected()
36     {
37         PlayFabClientAPI.GetUserdata(new GetUserdataRequest(), result =>
38         {
39             if (result.Data != null && result.Data.ContainsKey("currentCoinsCollected"))
40             {
41                 finalCoins = int.Parse(result.Data["currentCoinsCollected"].Value);
42                 DisplayFinalCoins();
43             }
44             else
45             {
46                 Debug.LogError("CurrentCoinsCollected data not found in PlayFab.");
47             }
48         }, OnError);
49     }
50 }
```

The first section of this code is the code connected to the 'Exit to Play Menu' button on the game summary screen. This code utilises Unity's Scene Manager in order to let the player go back to the Play menu after they have completed a game. This is essential, as it allows smooth navigation between menus and lets the user play the game repeatedly. This links to success criteria number 11, 'Game Summary UI', as this is the code that makes this specific button on the menu work.

The next section of this code is the declaration of variables and text game objects to display the final score and final amount of coins collected. This includes the declaration of 'finalScoreText' and 'finalCoinsText' as text game objects used to show the player their statistics, and 'finalCoins' and 'finalScore' to hold this data before it is shown to the user. This links to success criteria number 50, showing player statistics after death, because these are where the objects and variables used to do so are declared.

The next section of this code is the 'Start' subroutine, which is run by Unity the moment the menu is opened. This is beneficial, as it is used to immediately show the player their statistics, by calling the subroutines 'FetchCurrentCoinsCollected' and 'FetchCurrentScore'. The use of the start function, which is immediately called, means that the display is immediately shown when the player dies and is shown in this menu. This links to success criteria number 50, as the calling of these subroutines is what makes the game statistics be displayed to the player immediately after they die.

The final section of this code is the subroutine that fetches the coins collected in the most recent game, called 'FetchCurrentCoinsCollected'. This creates a new PlayFab get user data request, and stores the retrieved value in the 'finalCoins' subroutine, if the field exists and the value is not null. If the field does not exist or the value is null, then a message is output in the debug log, indicating this to the developer. The use of the debug log encourages maintainability in the system, and allows developers to see if there are any issues in the system anytime its run. If the retrieval of this data is unsuccessful for any reason, the 'OnError' subroutine is called, which outlines to the developer what the error is. This entire section of code contributes to the fulfillment of success criteria number 50, this is the code that retrieves the information to be able to be displayed to the user. This also links to success criteria number 11, as this coin text game object partly makes up the user interface for the game summary menu.

Lines 49 to 86 of Game Summary Code

```

1
49     void DisplayFinalCoins()
50     {
51         //displays the final coins collected on the summary
52         finalCoinsText.text = finalCoins + "";
53     }
54
55
56     //retrieves the current score from PlayFab & saves it to 'finalScore'
57     void FetchCurrentScore()
58     {
59         PlayFabClientAPI.GetUserDatavnew GetUserDataRequest(), result =>
60         {
61             if (result.Data != null && result.Data.ContainsKey("currentScore"))
62             {
63                 finalScore = float.Parse(result.Data["currentScore"].Value);
64                 DisplayFinalScore();
65             }
66             else
67             {
68                 Debug.LogError("CurrentScore data not found in PlayFab.");
69             }
70         }, OnError);
71     }
72
73     void DisplayFinalScore()
74     {
75         // Display the final score onscreen
76         finalScoreText.text = finalScore + "";
77     }
78
79     2 references
80     void OnError(PlayFabError error)
81     {
82         Debug.LogError("Error retrieving current coins collected from PlayFab: " + error.GenerateErrorReport());
83     }
84
85

```

The first section of this code is used to display the final coins collected after a game. This is done through assigning the 'finalCoins' variable to the 'finalCoinsText' game object, which is a UI element that can display the value to the user. This is done so that success criteria number 50 can be fulfilled, as after the information is retrieved from the external database, the coin information is displayed to the user using this subroutine, allowing them to see their coin statistics for the most recent game.

The next section of this code is the subroutine 'FetchCurrentScore', which is used to retrieve the score from the most recent game from the external database. This is necessary, as the Play Manager (game object that manages gameplay) sends the data from the current game to the external database after the player dies, so that it can be shown to the user. This subroutine first checks to see if the value in the field is null and if the field exists or not for this specific player. If the value is null or the field doesn't exist, then there is a debug log message shown to the developer to say that the current score was not found in PlayFab. If the field exists and there is a value (not null), then this is assigned to the 'finalScore' variable (this is so it can be later used to assign to the final score text game object). The subroutine 'DisplayFinalScore' is then called to display this to the user. If there is an error in the attempt to retrieve data from the external database, then the subroutine 'OnError' is called, to notify the developer of the issue. This links to success criteria number 50, as it is the retrieval of this data that allows it to be displayed to the user after their in-game death.

The last section of those code is 2 subroutines, one called 'DisplayFinalScore' and another called 'OnError'. 'DisplayFinalScore' is a subroutine that is called to display the final score to the user, through the use of the 'finalScoreText' game object. This is a UI text element that is shown to the player through the Unity Canvas game object, and so has to have the variable 'finalScore' assigned to it so that it can be displayed as text to the user. This links to success criteria number 50, as this (alongside the display of the final score) is what makes up the display of game statistics after the player dies. This also links to success criteria number 11, as this final score text game object partly makes up the user interface for the game summary menu.

The other subroutine detailed in this last section is the 'OnError' subroutine. This is a subroutine that runs if there is an issue in the retrieval of data from PlayFab. This is for the developer, and helps to keep the code maintainable, as the developer is always aware of any issues that arise (as they are written in the debug log).

5.3.3 - Game Summary Iterative Testing

5.3.3.1 - Test Table

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/Failure	Remedial Actions
Exit To Play Menu Button	67	Pressing the exit to play menu button	Player is taken to the Play Menu	Player is taken to the Play Menu	Success	N/A
	68	Pressing anywhere on the screen that is not a button	Nothing	Nothing	Success	N/A
Display of Final Score	69	Finishing a game	Final score displayed on screen	Final score displayed on screen	Success	N/A
Display of Final Coins	70	Finishing a game	Final Coins displayed on screen	Final Coins displayed on screen	Success	N/A

5.3.3.2 - Test Evidence

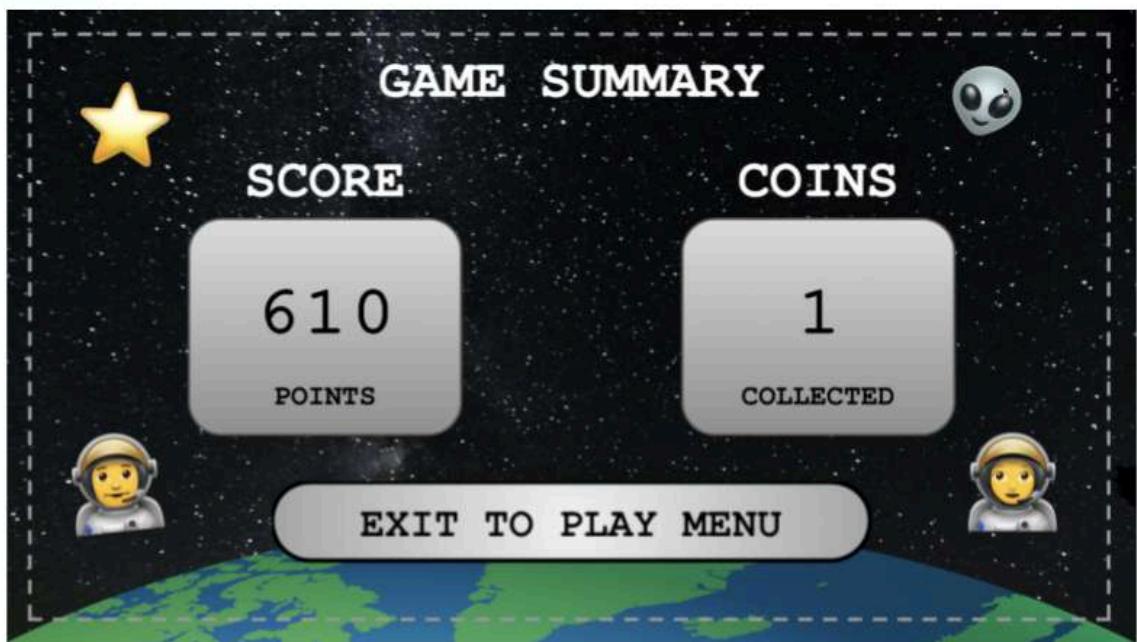
Test No. 67

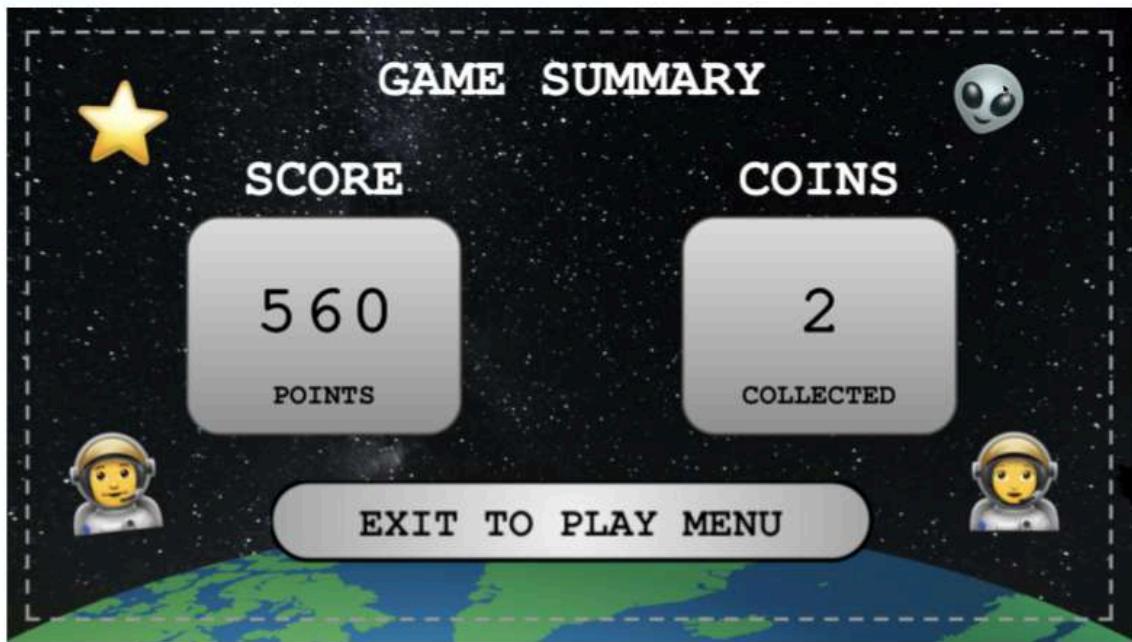


Test No. 68



Test No. 69





5.3.4 - Review of Game Summary Development

All of the tests for the game summary screen were successfully completed, with all of the outcomes being as expected. This means that this menu was successfully developed, and no further development or remedial actions need to be taken.

6 - AstroSprint Final Prototype (v4.0)

6.1 - Feedback on AstroSprint v4.0

6.1.1 - Primary User Introduction & Feedback on v4.0

In order to test my game, I have found a primary user, who fits my target audience, to play through my game and give their opinions on it, features they would like to improve, or see in future.

6.1.1.1 - Primary User Introduction

Name: Isabella Brown

Age: 17

Occupation: Student, part-time sales assistant

How she relates to the AstroSprint Audience: Isabella is a young adult within the age range of my stakeholders (13-18 year olds).

- Between her studies at college and her part-time job, she doesn't have lots of spare time to play games, but still likes to enjoy herself on her commute to work and school, as well as her downtime after long shifts.
- Isabella is a typical user of AstroSprint, as she needs a simple, entertaining game to play in her short periods of rest.

Signature:

A handwritten signature in black ink, appearing to read "Isabella Brown".

6.1.1.2 - Primary User Feedback

Evidence for primary user playthrough is in the attached video: '3002_RichelleAmaAcheampong_H446-03_PrimaryUserFullGameplayV1'.

After playing through the original prototype, she has given the following feedback on what the next prototype should change, fix and improve:

- Isabella detailed about how the menu label 'Tutorial' was misleading, as she believed that it implied there would be a guided walk through of the game, rather than just a screen that showed the controls
- Isabella questioned why the hard mode button is present if the option itself hasn't been developed, so suggested the removal of it and making easy mode the only mode
 - I then asked her how the menu layout should be changed, and she suggested that the label should change and say "like 'Play' or 'Endless' or something, up to you" (direct quote)

- She also detailed about how it is unfair that the user flies off the screen and dies when using the 'flying' power up, so that should be changed so either the player doesn't die or they can't leave the screen
- She also mentioned that there should be some sort of indication of how long a power up has left of being active, as she could not tell how long she had to use a power up before it ran out

6.1.2 - Developer Feedback on v4.0

6.1.2.1 - Video Evidence

- Evidence for primary user playthrough is in the attached video: '3002_RichelleAmaAcheampong_H446-03_PrimaryUserFullGameplayV3'
- Evidence for play through of pause menu is on '3002_RichelleAmaAcheampong_H446-03_PauseMenuV3'
- Evidence for play through of abilities menu is on '3002_RichelleAmaAcheampong_H446-03_AbilitiesV3'
- Evidence for play through of easy mode is on '3002_RichelleAmaAcheampong_H446-03_DeveloperGameplayV3'
- Evidence for play through of flying power up is on '3002_RichelleAmaAcheampong_H446-03_FlyingV3'

6.1.2.2 - Developer Issues

After seeing the primary user play through AstroSprint v3.0, playing through the biggest issues (the abilities menu and the pause menu) and reading through primary user feedback I have the following issues with AstroSprint v3.0:

- Obstacles spawn too often
 - Having too many obstacles on screen, as shown in the primary user gameplay video and developer gameplay video, makes the game a lot more difficult and much more cluttered
 - It also makes it extremely difficult to collect power ups and coins, as they often spawn near the obstacles
- Obstacles falling physics stops working after collision with foreground
 - As shown in the developer gameplay video, the obstacles just float in position and stop falling after they have collided with the ground at least once, and only begin to move again (at a slower speed) if they collide with another object
 - Even after the foreground moves to have a lower height, the obstacle does not continue to fall to the ground.
- Power ups spawn too often
 - Having too many power ups on screen, as shown in the primary user gameplay video and the developer gameplay video, makes the game a lot easier to beat and much more cluttered
 - Having too many power ups also makes the game too easy, as they are made to make the game easier
- Double Jump power up still active after death of character and a new game beginning

- As shown in video 4, after the player dies in the game, exits to the play menu and returns to the game, the game continues to be paused.
- This is an issue, because it means the player is unable to play the game if they quit the previous game with the button
- Foreground not correlated to Space Theme
 - The foreground of the game (as shown in the developer gameplay video, video number 4) is just black, and doesn't relate to the overall theme of space
- Background not correlated to Space Theme
 - Similar to the foreground, the background is just a gradient of colour which doesn't relate to space.
 - This was very evident in the gameplay video for the primary user and the developer, which shows the background of the game in Easy mode.
- Pause Menu Bugs (All shown in Pause Menu video):
 - 2 Clicks to get to the pause menu: The game doesn't take the user to the pause menu after pausing, it just pauses all of the functions of the game.
 - Pausing leaving & rejoining keeps game paused: As shown in video 2, leaving the game through the pause menu and playing again means that the user can't restart the game through this method
 - Tutorial image doesn't hide when button clicked again
- Issues with Flying (All issues shown in Flying V1 video)
 - Obstacles spawn at the same height when flying active, making the game less challenging when this power up is active, as you can simply fly over the spawn point of the obstacles (the middle of the screen).
 - Flying too high up causes the player to go offscreen, and therefore die and the game to be over.
 - This is not fair, as it means that the power up has a massive drawback
- Different fonts for sign up menu compared to the rest of the game
 - As shown in the primary user gameplay video, when the primary user is signing up, the sign up menu has different font to the rest of the game, as it uses 'Liberation Sans' font instead of 'Courier New'
 - This is not good, as it makes the game less cohesive
- Abilities Menu (As shown in the abilities v1 video)
 - As shown in the abilities menu video, the coin balance display not updated after a purchase, so player cannot see how many coins they have left after a purchase
 - Also shown in that video, the maximum upgrade is 1 less than it should be, it fills in a maximum of 4 bars, rather than 5. This means users are unable to fully upgrade any power up because the upgrade button is disabled prematurely
 - Another issue is the other update buttons being disabled temporarily after the 1 upgrade has reached maximum level, before allowing the purchasing of power ups again

6.1.2.3 - Shortlisted Features To Be Implemented

Due to limitations in my knowledge of programming and the fact that I am a single person working on this code with a time limit, I have decided to prioritise these aspects in AstroSprint v4.0:

- Change background design to relate to the space theme
- Power ups spawning less often
- Obstacles spawning less often
- Pause Menu stopping time not carrying over into the next game
- Make the obstacles continue to fall even after coming into contact with the foreground
- Make the obstacles spawn from a higher height
- Make obstacles fall faster
- Change the font on 'Sign Up' input fields so that it remains consistent to the game
- Getting rid of hard mode
- Renaming Easy Mode
- Change arrangement of buttons on Play Menu
- Updating coin balance after every purchase

6.2 - AstroSprint v4.0 Success Criteria

In addition to the success criteria created in the analysis stage (found in [1.5 - Success Criteria](#)), these are the additional success criteria to be in the final prototype of AstroSprint, called AstroSprint v4.0:

Success Criteria	Criteria No.	Justification	How is it Measurable?
Change background design to relate to the space theme	1	Current background is just a gradient of colour which doesn't relate to space, which needs to change to something more space related to make the game more cohesive	Screenshot of previous background and the new background, with detail on what has changed
Change the font on 'Sign Up' input fields so that it remains consistent to the game	2	The AstroSprint v3.0 font of the 'Sign Up' menu is not the same as the rest of the game, which needs to be changed	Screenshot of Unity Inspector showing old font, and then showing the change to the new font Screenshot of the sign up menu input

		to make the whole game more cohesive	boxes having the 'Courier New' font
Power ups spawning less often	3	The short spawn interval in AstroSprint v3.0 means there are too many power ups on screen at any given time. This makes the game a lot easier to beat and much more cluttered on screen. This needs to be changed so they spawn less often	Screenshot of old code that declared the spawn interval of power ups, and screenshot of new code with spawn interval changed
Obstacles spawning less often	4	The short spawn interval in AstroSprint v3.0 means there are too many obstacles on screen at any given time. This makes the game a lot harder to beat and much more cluttered on screen. This needs to be changed so they spawn less often.	Screenshot of old code that declared the spawn interval of obstacles, and screenshot of new code with spawn interval changed
Pause Menu stopping time not carrying over into the next game	5	Necessary to ensure a user can exit the game through the play menu and begin to play again, without the game beginning as being paused.	Screenshot of code, highlighting new parts that ensure that this doesn't happen
Obstacles continue to fall even after coming into contact with the foreground	6	Necessary so that obstacles are on the ground at all times, even if the foreground changes height	Screenshot of code, highlighting new parts that ensure that this happens

Obstacles spawn from a higher height	7	To ensure that the game remains challenging no matter what power ups are active	Screenshot of old code that declared the spawn height of obstacles, and screenshot of new code with spawn height changed
Obstacles fall faster	8	Necessary so that they can be settled on the ground before the user gets to them so it is easier for them to try and avoid them	Screenshot of old code that declared the fall speed of obstacles, and screenshot of new code with fall speed changed
Removal of hard mode option	9	As I do not have enough time to create this game mode, the removal of this as an option means that the user will not be misled with what their options for playing are	Screenshots of Play Menu code, highlighting what is to be removed and screenshots of code with those sections removed
Renaming Easy Mode to 'Play'	10	Necessary as the hard mode option is being removed, so there is only one option, to 'Play'	Screenshot of old vs new button design Screenshot of Play Menu with this button renamed
Rearrangement of Play Menu to reflect game mode changes	11	Necessary so that the menu doesn't have any gaps and remains looking cohesive	Screenshot of old Play Menu design, and new Play Menu design
Updating coin balance after every purchase	12	Necessary so that the player knows how many coins they have after every purchase, not just when the menu is loaded	Screenshot of new code including these changes, highlighting the code that has been added
Renaming of Tutorial label to 'Controls'	13	Necessary so that the name of the menu is not misleading,	Screenshot of old Play Menu design, and new Play Menu

		because it is not an interactive tutorial, just a screen	design Screenshot of old Pause Menu design, and new Pause Menu design
--	--	--	--

6.3 - AstroSprint v4.0 Development

6.3.1 - AstroSprint v4.0 Changes

'S.C.' Refers to the success criteria, where 'SC No. 1' refers to the first item in the success criteria.

6.3.1.1 - Changing Background (S.C. No. 1)

The background of AstroSprint v3.0 is just gradient of colour:



I have decided on making the gradient darker, so it seems more like space. I have also decided to add stars to the background in order to make it clear the game is set in space.

Both this new background and the original background's colour gradient were created on <https://colors.co/>.

The stars overlaid on the background come from Vecteezy.com: (Link: <https://www.vecteezy.com/png/16586629-twinkle-star-pattern-for-photo-effect-and-overlay-abstract-blurry-star-light-texture-for-background>)

The final design for the background is as follows:

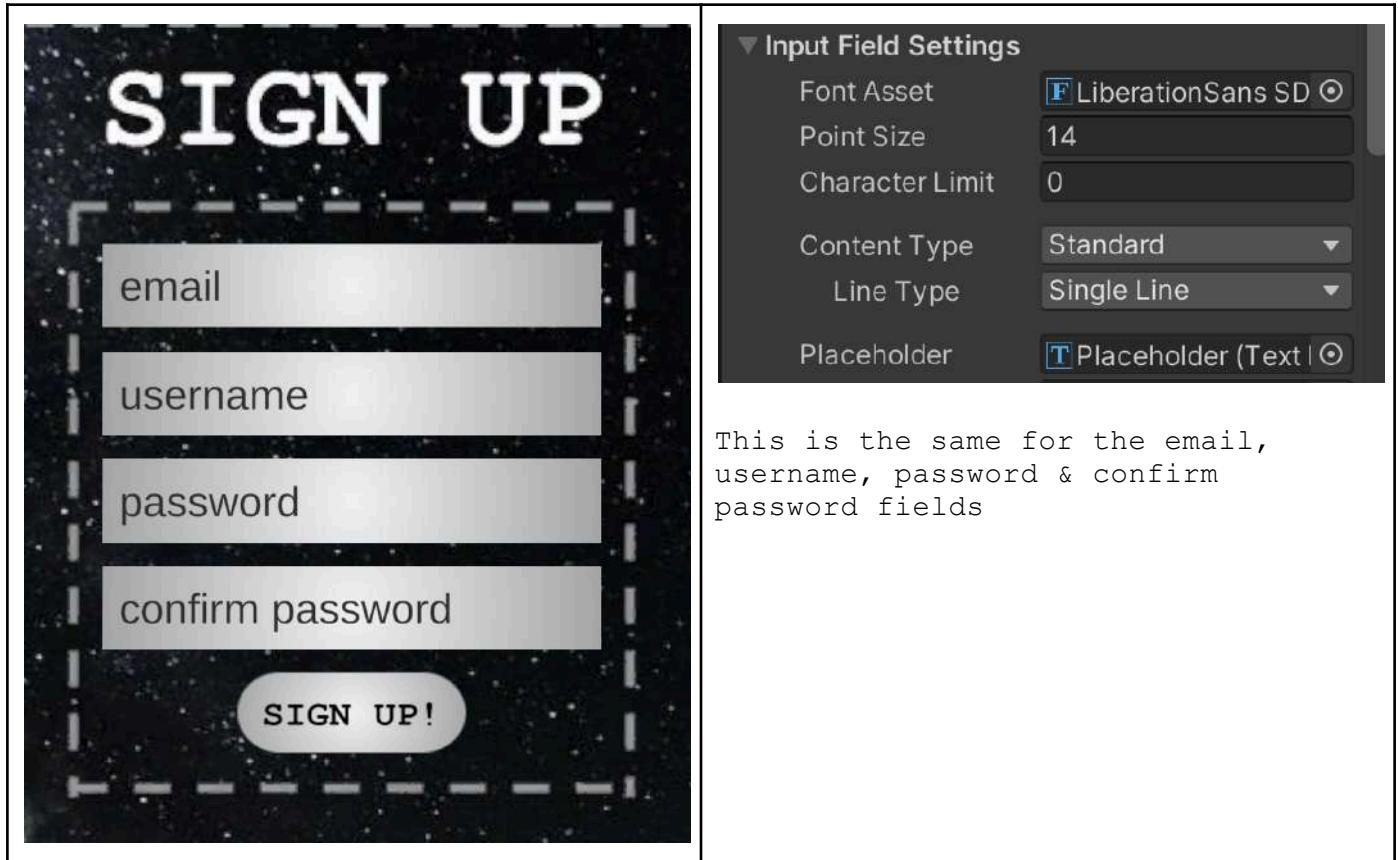


This new background has stars, and a new colour scheme in the background. The colour has been changed to this colour to make it look less like earth, as the previous blue sky made it look like the game was set on Earth, which it should not.

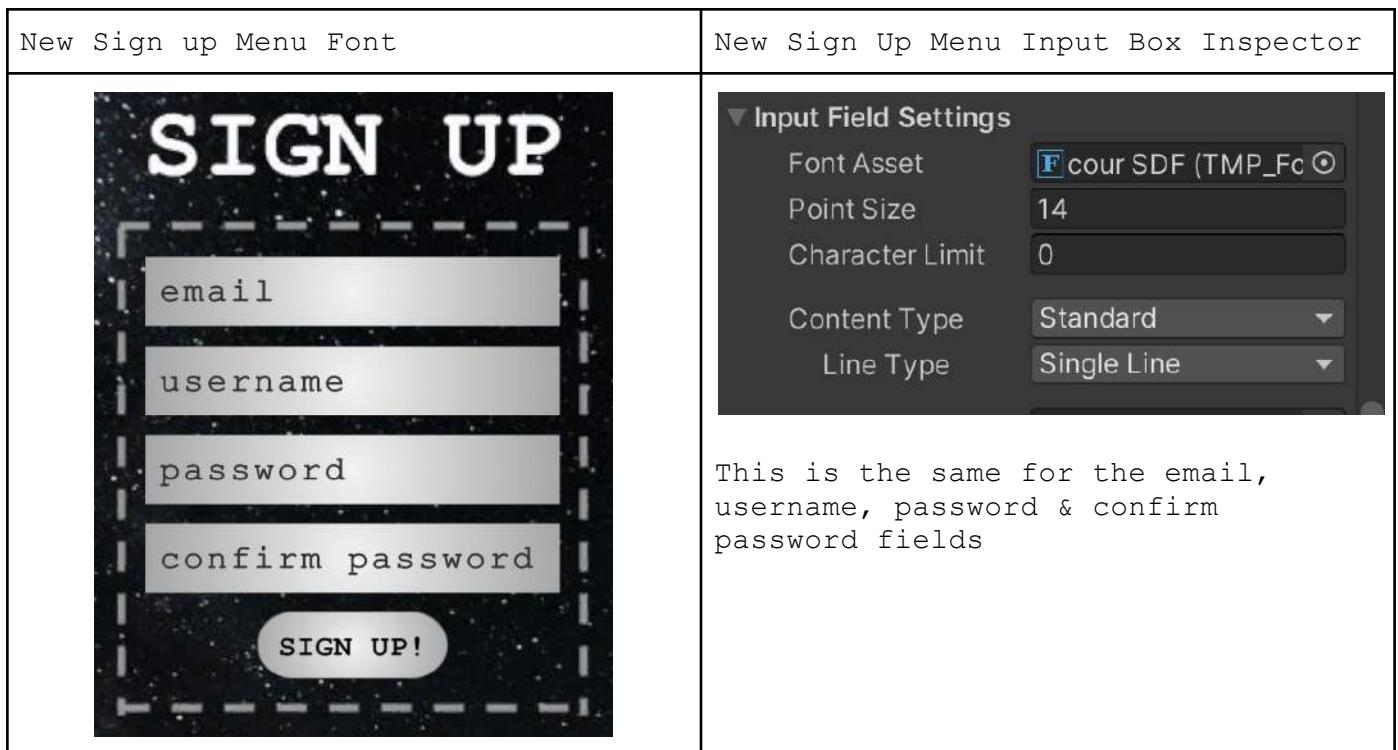
6.3.1.2 - Sign Up Menu Font (S.C. No. 2)

In order for there to be cohesion in the menus, the sign menu must use the same font as that of the rest of the program

Old Sign up Menu Font	Old Sign Up Menu Input Box Inspector
-----------------------	--------------------------------------



I have assigned the 'Courier New' SDF file (how Unity TMP Text game objects accept fonts) to all of the fields in the sign up menu, so now it is as shown:



6.3.1.3 - Power Up Spawn Interval (S.C. No. 3)

This is the code from AstroSprint v4.0 that declares the spawn interval for all power ups. In my final iteration, AstroSprint v5.0, I am changing this to be a larger value, so that there is a bigger time between spawns. This means less power ups are spawned in a given time frame.

This is the old code for the spawn interval declaration:

[Line 7 on 'Power Up Spawner' code from AstroSprint v4.0]

```
    1 reference  
7 |  public float spawnInterval = 7f; //interval between spawns  
    1 reference
```

I am going to change this to a higher number, 14, so that it takes twice as long to spawn a power up. I have changed the code for AstroSprint v5.0 as shown below:

```
    1 reference  
7 |  public float spawnInterval = 14f; //interval between spawns  
    1 reference
```

6.3.1.4 - Obstacle Spawn Interval (S.C. No. 4)

This is the code from AstroSprint v4.0 that declares the spawn interval for all obstacles. In my final iteration, AstroSprint v5.0, I am changing this to be a larger value, so that there is a bigger time between spawns. This means less power ups are spawned in a given time frame.

This is the old code for the spawn interval declaration:

[Line 8 on 'Obstacle Spawner' code from AstroSprint v4.0]

```
    1 reference  
8 |  public float spawnInterval = 7f; //time interval between spawn  
    1 reference
```

I am going to change this to a higher number, 12, so that it takes twice as long to spawn an obstacle. I have changed the code for AstroSprint v5.0 as shown below:

```
    1 reference  
8 |  public float spawnInterval = 12f; //time interval between spawn  
    1 reference
```

6.3.1.5 - Changing Pause Menu Function (S.C. No. 5)

This is the original code that controls the function of the 'Exit To Play Menu' Button from AstroSprint v4.0:

[Lines 52 to 57 of 'Pause Menu' Code, AstroSprint v4.0]

```
0 references
52     public void ExitToPlayMenu()
53     {
54         //loads the play menu scene
55         SceneManager.LoadScene("Play Menu");
56     }
57 }
```

As shown above, the only function of this menu is to change the scene, and doesn't reverse the effects of the function ('Pause', shown below) that stops the progression of time in the game.

[Line 35 to 39 of 'Pause Menu' Code, AstroSprint v4.0]

```
1 reference
35     public void Pause()
36     {
37         pauseMenuUI.SetActive(true);
38         Time.timeScale = 0f; //pause gameplay by freezing time
39     }
40 }
```

In order to make sure the player is able to play from an unpause state after quitting, the 'ExitToPlayMenu' must contain code that makes the game return to the original time scale. In AstroSprint v5.0, I have added code that does this:

```
0 references
52     public void ExitToPlayMenu()
53     {
54         Time.timeScale = 1f; //makes time return to its normal progression
55         SceneManager.LoadScene("Play Menu"); //loads the play menu scene
56     }
57 }
58 }
```

Line 54 is the new line that achieves this, as it unfreezes time and makes it return to its normal scale. This should make sure that the pause from the previous game doesn't affect the new one.

6.3.1.6 - Obstacle Falling Physics (S.C. No. 6)

This is the code from AstroSprint v3.0 that controls whether the obstacles keep falling after a collision with the ground.

```
40
41         //stop the obstacle's movement when it hits the ground
42         if (collision.gameObject.CompareTag("Ground"))
43         {
44             rb.velocity = Vector2.zero;
45         }
46     }
47 }
```

The code below explicitly (Line 43) makes sure that the obstacle stops moving if it collides with the ground. In order to make sure it keeps falling as long as it is not on the ground, I am going to remove this section of code.

In the AstroSprint v3.0 code, the entire collision subroutine is this:

```
0 references
33 void OnCollisionEnter2D(Collision2D collision)
34 {
35     //destroy the obstacle if it collides with the player
36     if (collision.gameObject.CompareTag("Player"))
37     {
38         Destroy(gameObject);
39     }
40
41     //stop the obstacle's movement when it hits the ground
42     if (collision.gameObject.CompareTag("Ground"))
43     {
44         rb.velocity = Vector2.zero;
45     }
46 }
47 }
```

However, after the removal of lines 41 to 45, the subroutine will become this in AstroSprint v4.0:

```
    0 references
31     void OnCollisionEnter2D(Collision2D collision)
32     {
33         if (collision.gameObject.CompareTag("Player"))
34         {
35             Destroy(gameObject); //destroy the power up on player collection
36         }
37     }
38 }
39 }
```

This code should be able to allow obstacles to remain falling.

6.3.1.7 - Obstacle Spawn Height (S.C. No. 7)

This is the code from AstroSprint v3.0 that declares the spawn height for all obstacles. In my final iteration, AstroSprint v4.0, I am changing this to be a larger value, so that the obstacles spawn higher up on screen.

This is the old code for the spawn height declaration, the yPosition variable, AstroSprint v3.0:

```
    1 reference
9     public float yPosition = -0.46f; //y position for obstacles spawning
10
```

I am going to change this to a higher number, -0.2, so that it falls at twice the speed in v5.0 as it did in v4.0. I have changed the code for AstroSprint v5.0 as shown below:

[From AstroSprint v4.0]

```
    1 reference
9     public float yPosition = -0.2f; //y position for obstacles spawning
10
```

After this change, the obstacles should fall at a faster speed.

6.3.1.8 - Obstacle Fall Speed (S.C. No. 8)

This is the code from AstroSprint v3.0 that declares the fall speed for all obstacles. In my final iteration, AstroSprint v4.0, I am changing this to be a larger value, so that the obstacles fall faster on screen.

This is the old code for the fall speed declaration, AstroSprint v3.0:

```
    1 reference
8     public float fallSpeed = 4f; // speed that the obstacle falls
    1 reference
```

I am going to change this to a higher number, 8, so that it falls at twice the speed in v5.0 as it did in v4.0. I have changed the code for AstroSprint v4.0 as shown below:

[From AstroSprint v4.0]

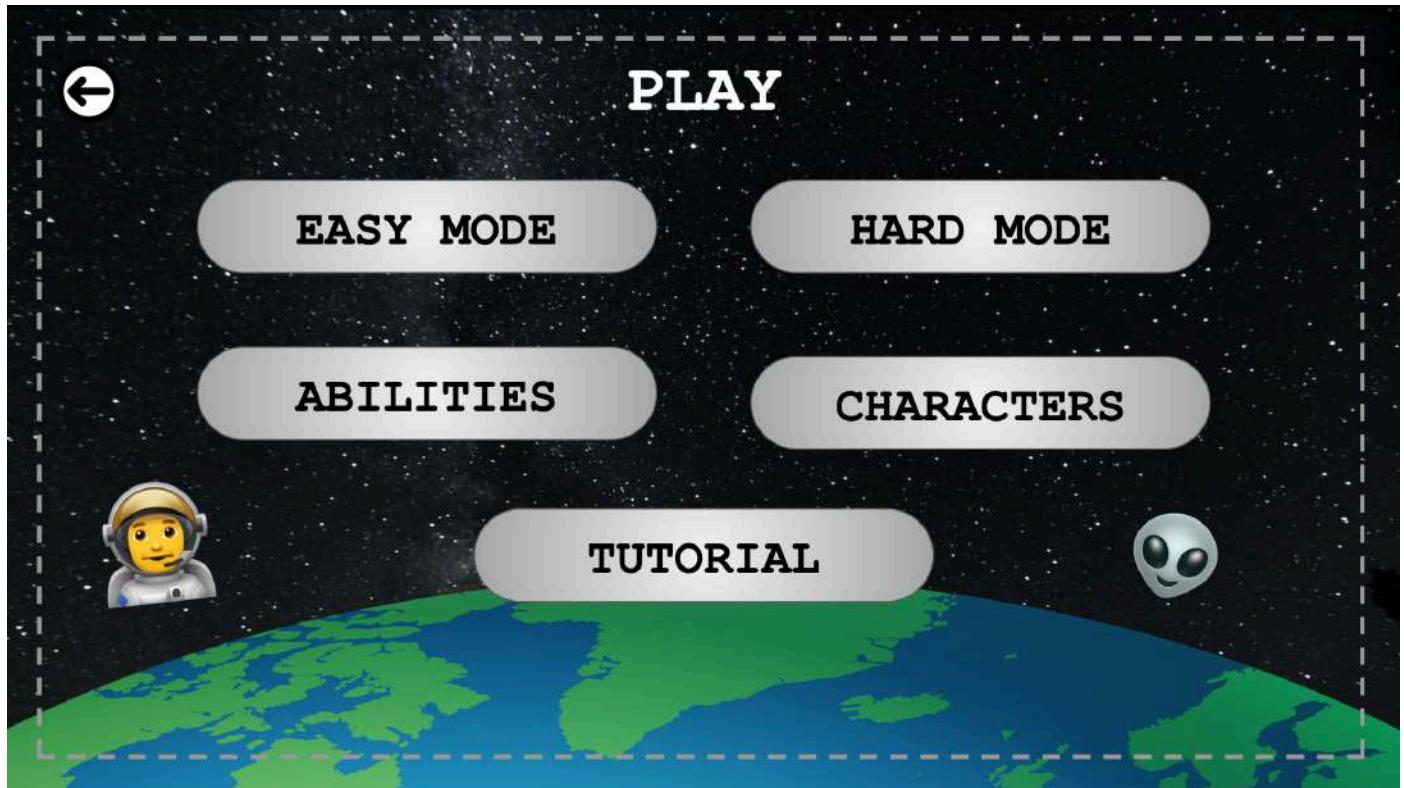
```
1 reference  
8 | public float fallSpeed = 8f; // speed that the obstacle falls  
1 reference
```

After this change, the obstacles should fall at a faster speed.

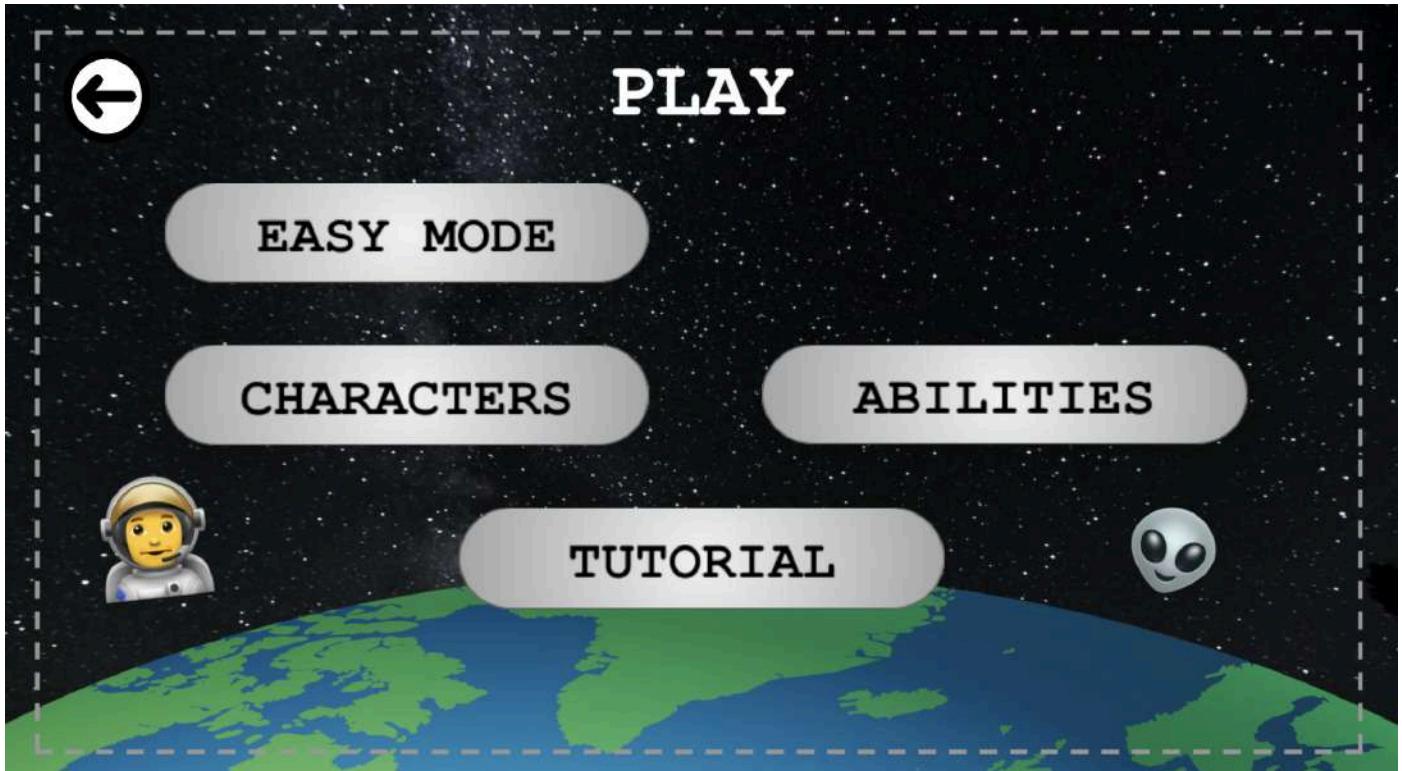
6.3.1.9 - Removal of Hard Mode (S.C. No. 9)

This is the original Play Menu before the hard mode option was removed:

[From AstroSprint v3.0]



For AstroSprint v4.0, I am removing the hard mode option, so after this change, the Play Menu will look like this:



And the removal of all code associated with the hard mode, which is as follows (taken from AstroSprint v3.0):

```
26  
27     //takes user to hard mode gameplay  
28     0 references  
29     public void GoToHardMode()  
30     {  
31         SceneManager.LoadScene("Hard Mode");  
32     }
```

6.3.1.10 - Renaming of Easy Mode (S.C, No. 10)

In order to reflect the idea that there is only 1 game mode, I am changing the play menu to say 'Play' rather than 'Easy Mode'. This is shown below:



6.3.1.11 - Rearrangement of Play Menu (S.C. No. 11)

Due to there being a large gap in the Play Menu, the Play Menu has to be rearranged to fill in the gap, and make the menu look more cohesive. The current menu (as shown on the right) is not cohesive, and that is not beneficial for usability.



This means that I am going to rearrange the menu so that it doesn't have a large gap and is cohesive. This was done as follows:



This is the final arrangement of the Play Menu, to be used in AstroSprint v4.0.

6.3.1.12 - Frequency of Coin Balance (S.C. No. 12)

This is the original code from AstroSprint v3.0 that updates the coin balance text:

```
123     // updates the text element to display current coin balance
124     3 references
125     private void UpdateCoinBalanceText()
126     {
127         coinBalanceText.text = coinBalance.ToString();
128     }
```

I will be keeping this snippet of code for AstroSprint v4.0, but for this final iteration, I will call this subroutine more often. I will call the subroutine in every instance where there's anything to do with power ups and pricing, in an effort to show the updated coin balance every single time the user makes a purchase. Calling this subroutine more often means that the coin balance display is updated as much as possible.

The new code is below, with new aspects highlighted with a red box around it:

```

172     // subroutine to update a specific power up
173     0 references
174     public void TryUpgradePowerUp(int powerUpIndex)
175     {
176         if (powerUpLevels[powerUpIndex] < prices.Length && coinBalance >= prices[powerUpLevels[powerUpIndex]])
177         {
178             // deducts the price of the upgrade from balance
179             coinBalance -= prices[powerUpLevels[powerUpIndex]];
180             UpdateCoinBalanceText()
181
182             // updates & displays the power up level & saves the new coin balance & power up levels
183             powerUpLevels[powerUpIndex]++;
184             SaveCoinBalance();
185             SavePowerUpLevels();
186             UpdateUI();
187         }
188         else
189         {
190             feedbackText.text = "You don't have enough coins for this upgrade";
191         }
192     }
193 }
```

```

2 references
82     private void OnSaveUserDataSuccess(UpdateUserDataResult result)
83     {
84         UpdateCoinBalanceText()
85         Debug.Log("Coin balance successfully updated in PlayFab.");
86     }
```

6.3.1.13 - Renaming of Tutorial Screen (S.C. No. 13)

In order to make my game more usable and make the menu options be less misleading, I have decided to change the name of the 'Tutorial' screen to say 'Controls', as that more accurately reflects the contents of this screen.

This means that this label has to be changed on the buttons on both the Play Menu and the Pause Menu. This also means that the title at the top of the screen needs to be changed to say 'Controls'.

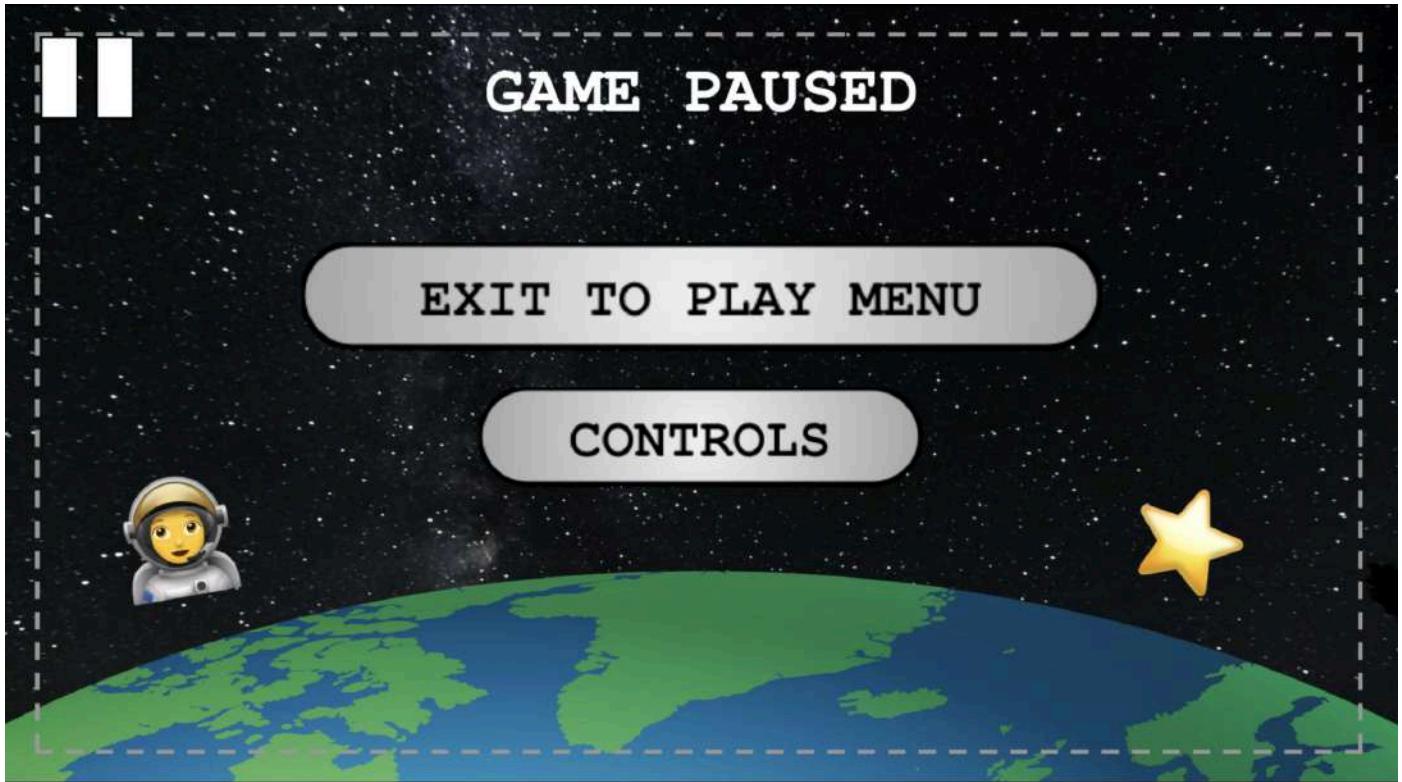
Controls Screen [AstroSprint v4.0]:



Play Menu [AstroSprint v5.0]:



Pause Menu [AstroSprint v4.0]:



6.3.2 - AstroSprint v4.0 Testing

6.3.2.3 - Test Planning

The 'Success Criteria' column refers to the success criteria made in [4.2 - AstroSprint v4.0 Success Criteria](#) and its corresponding numbers, not the success criteria made in [1.5 - Success Criteria](#).

Feature to Test	Success Criteria No.	Test No.	Test Data to be Used	Justification of Test Data	Expected Outcome
Power up spawn interval	3	1	Playing the actual game by clicking 'Play'	The only way to see if the power ups are spawning less often is to play AstroSprint v3.0, and compare to the play through of AstroSprint v4.0	Power ups spawn less often in AstroSprint v4.0 as compared to AstroSprint v3.0
Obstacles spawn interval	4	2	Playing the actual game by clicking 'Play'	The only way to see if the obstacles are spawning less often is to play AstroSprint v3.0, and compare to the play through of	Obstacles spawn less often in AstroSprint v4.0 as compared to AstroSprint

				AstroSprint v4.0	v3.0
Pause Menu stopping time not carrying over into the next game	5	3	Playing actual game, exiting to the play menu through the pause menu, and clicking on the 'play' button to play another game	The only way to see if this works is by going through the steps (described in test data) and see if the player is able to play from an unpause state if they quit the game through the pause menu. The steps described in the Test Data column are the	Player is able to leave the game through the pause menu, play the game again, and begin with the game unpause
Obstacles continue to fall even after coming into contact with the foreground	6	4	Playing the actual game by clicking 'Play'	The only way to see if the obstacles continue to fall after making contact with the ground is to play AstroSprint v4.0, and see if they do or not.	Obstacles continue to fall when the height of the foreground changes, even after contact with the foreground
Obstacle Spawn Height	7	5	Playing the actual game by clicking 'Play'	The only way to see if the obstacles are spawning at a higher height is to play AstroSprint v3.0, and compare to the play through of AstroSprint v4.0	Obstacles spawn at a higher in AstroSprint v4.0 as compared to AstroSprint v3.0
Obstacles fall speed	8	6	Playing the actual game by clicking 'Play'	The only way to see if the obstacles are falling faster is to play AstroSprint v3.0, and compare to the play through of AstroSprint v4.0	Obstacles fall faster in AstroSprint v4.0 as compared to AstroSprint v3.0

6.3.2.2 - Test Table

Feature to Test	Test No.	Test Data	Expected Outcome	Actual Outcome	Success/ Failure	Remedial Actions
Power up spawn interval	1	Playing the actual game by clicking 'Play'	Power ups spawn less often in AstroSprint	Power ups spawn less often in AstroSprint	Success	N/A

			v4.0 as compared to AstroSprint v3.0	v4.0 as compared to AstroSprint v3.0		
Obstacles spawn interval	2	Playing the actual game by clicking 'Play'	Obstacles spawn less often in AstroSprint v4.0 as compared to AstroSprint v3.0	Obstacles spawn less often in AstroSprint v4.0 as compared to AstroSprint v3.0	Success	N/A
Pause Menu stopping time not carrying over into the next game	3	Playing actual game, exiting to the play menu through the pause menu, and clicking on the 'play' button to play another game	Player is able to leave the game through the pause menu, play the game again, and begin with the game unpause	Player is able to leave the game through the pause menu, play the game again, and begin with the game unpause	Success	N/A
Obstacles continue to fall even after coming into contact with the foreground	4	Playing the actual game by clicking 'Play'	Obstacles continue to fall when the height of the foreground changes, even after contact with the foreground	Obstacles continue to fall when the height of the foreground changes, even after contact with the foreground	Success	N/A
Obstacle Spawn Height	5	Playing the actual game by clicking 'Play'	Obstacles spawn at a higher in AstroSprint v4.0 as compared to AstroSprint v3.0	The obstacles continued to spawn at the same height they do in AstroSprint v3.0	Failure	Re-evaluate all of the code from the Obstacles and try to find why this may not work
Obstacles fall speed	6	Playing the actual game by clicking 'Play'	Obstacles fall faster in AstroSprint v4.0 as	Obstacles fall faster in AstroSprint v4.0 as	Success	N/A

			compared to AstroSprint v3.0	compared to AstroSprint v3.0		
--	--	--	------------------------------------	------------------------------------	--	--

6.3.2.3 - Test Evidence

Evidence of Tests 1, 2, 5 & 6 can be found on the video titled '3002_RichelleAmaAcheampong_H446-03_PrototypeComparisonVideo'

Evidence of Tests 3 & 4 can be found on the video titled '3002_RichelleAmaAcheampong_H446-03_DeveloperGameplayFINAL'

6.3.3 - Review of AstroSprint v4.0 Development

There was only 1 test that failed in the development of AstroSprint v4.0, which is test number 5. This was a failure because the obstacles spawned at the same height in AstroSprint v3.0 and v4.0.

This could be rectified through analysing the obstacle code script, and trying to find how to spawn the obstacles higher up. This could potentially also be done through taking aspects from the coin spawner code or the power up spawner code, as both of these objects spawn from higher height than the obstacles.

Due to lack of time, this issue will not be rectified within my project, but is something I would endeavour to rectify if I had more time to develop.

7 - Evaluation

7.1 - Post Development Testing

**ALL EVIDENCE FOR POST DEVELOPMENT TESTING (APART FROM EXTERNAL DATABASE TESTS)
ARE WITHIN THE VIDEO TITLES
'3002_RichelleAmaAcheampong_H446-03_FinalIterationVideo'**

Test No.	Method of Testing	Expected Outcome	Actual Outcome	Successful, Partially Successful or Failure?
1	Go through of all menus with the title shown	Every single menu is titled	Every single menu is titled	Successful
2	Go through the full game, showing all of the menu titles and the options within them, like buttons leading to clear options	Every menu is clearly labelled with the options within them being clear	The play menu was not as clearly labelled as it could be. The rest of the menus are clearly labelled though.	Partially Successful
3	Video of main menu and all interactable features	The main menu has an interactable user interface, which is descriptive and fit for purpose	The main menu has an interactable user interface, which is descriptive and fit for purpose	Successful
4	Video of login menu and all interactable features	The login menu has an interactable user interface, which is descriptive and fit for purpose	The login menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
5	Video of sign up menu and all interactable features	The sign up menu has an interactable user interface, which is descriptive and fit for purpose	The sign up menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
6	Video of play menu and all interactable features	The play menu has an interactable user interface, which is	The play menu has an interactable user interface, which is descriptive and fit for	Fully Successful

		descriptive and fit for purpose	purpose	
7	Video of tutorial screen and all interactable features	The tutorial screen menu has an interactable user interface, which is descriptive and fit for purpose	Tutorial screen turned into a controls screen on later iterations, and is not a tutorial as it is not interactive, but it still guides the player on how to play the game	Partially Successful
8	Video of character menu and all interactable features	The character menu has an interactable user interface, which is descriptive and fit for purpose	The character menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
9	Video of abilities menu and all interactable features	The abilities menu has an interactable user interface, which is descriptive and fit for purpose	The abilities menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
10	Video of pause menu and all interactable features	The pause menu has an interactable user interface, which is descriptive and fit for purpose	The pause menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
11	Video of game summary menu and all interactable features	The game summary menu has an interactable user interface, which is descriptive and fit for purpose	The game summary menu has an interactable user interface, which is descriptive and fit for purpose	Fully Successful
12	Video showing use of back buttons on all menus	There is a back button present on every single menu, and it will take user back to the previous menu	There is no back button on the main menu to be able to exit the game. The rest of the menus have working back buttons though.	Partially Successful
13	Primary user shown video of gameplay and review	The entire colour scheme of the game was cohesive	The entire colour scheme of the game was cohesive	Fully Successful
14	Video of user playing in all of the different environments	There are varied environments in the game that the player can play in	Varied environments was not implemented into my game, and so could not be tested	N/A

15	Video of login menu being used, and a pre-existing user being able to log in.	There is a login system that allows pre-existing users to log in, and has input validation for all fields	There is a login system that allows pre-existing users to log in, and has input validation for all fields	Fully Successful
16	Video of sign up menu being used and new user being able to sign up	There is a login system that allows new users to sign up, and has input validation for all fields	There is a login system that allows new users to sign up, and has input validation for all fields	Fully Successful
17	Video of character menu being used, showing all options for characters within them	All character icons are visible and able to be selected	All character icons are visible and able to be selected	Fully Successful
18	Video of invalid and valid inputs being input into the email field and corresponding error messages	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	Fully Successful
19	Video of invalid and valid inputs being input into the username field and corresponding error messages	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	Fully Successful
20	Video of invalid and valid inputs being input into the password field and corresponding error messages	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	All Invalid inputs are met with the appropriate error messages are shown to the user. Valid inputs allow the user to sign up	Fully Successful
21	Video of passwords that match and don't match, and corresponding error messages for passwords that don't match	Password fields that match are allowed to sign up. Password fields that don't match cause an appropriate error message to appear.	Password fields that match are allowed to sign up. Password fields that don't match cause an appropriate error message to appear.	Fully Successful
22	Video of coin	Non-zero coin	Non-zero coin balance	Fully

	balance being shown on-screen from a pre-existing user that just logged in	balance is shown on-screen after a pre-existing user logs in	is shown on-screen after a pre-existing user logs in	Successful
23	Video of user upgrading power ups using coins and screenshot of the coin balance in external database after these purchases have been made	Coin balance decreases after a purchase is made	Coin balance decreases after a purchase is made	Fully Successful
24	Video of a character being selected and its corresponding indicator showing an image of that character.	The icon of the character selected is shown separately on the screen	The icon of the character selected is shown separately on the screen	Fully Successful
25	Video of user opening the abilities menu, and being able to see, on inspection, something that shows the power up's level	All of the power ups have a clear indication of how much they have been upgraded	All of the power ups have a clear indication of how much they have been upgraded	Fully Successful
26	Video of a new user opening the character menu and there being an indicator showing their currently selected character, before they have selected anything themselves.	The male astronaut character is selected even before any buttons are clicked.	The male astronaut character is selected even before any buttons are clicked.	Fully Successful
27	Video of challenges menu showing all of the different challenges, how	The challenges menu accurately and understandable displays all of the information	This menu was not implemented into my game, and so could not be tested	N/A

	much currency they are worth on completion, and the current progress towards completing the challenge	about the in-game challenges		
28	Video of every single menu and all of the gameplay and inspection on what planes the game exists on	The entire game is 2D	The entire game is 2D	Fully Successful
29	Video of easy mode gameplay, showing the horizontal movement getting progressively faster	There is a slow escalation of difficulty within the game	There is a slow escalation of difficulty within the game	Fully Successful
30	Video of gameplay, where there is a collision with a power up icon, with text shown at the top of the screen, naming the power up.	Collision with a power up icon causes the name of the power up to be shown at the top of the screen	If there is one power up active, this works. However, when there is more than one power up active, the name of the second power up replaces the name of the first power up, so they are not visible at the same time. Only 1 power up name is shown at once.	Partially Successful
31	Video of gameplay, where there is a collision with a power up icon, with a visual indicator showing the duration of the power up.	Duration of power ups indicated on-screen	Visible representations of power up durations were not implemented into my game, and so could not be tested	N/A
32	Video of gameplay, showing the use of power ups	Power up effects are able to be used in-game	Power up effects are able to be used in-game	Fully Successful
33	Video of upgrading the power ups on the	Power ups upgraded to level 1 last for 30 seconds,	Power ups upgraded to level 1 last for 30 seconds, level 2 for 40	Fully Successful

	abilities menu, and then going to the gameplay and timing how long it takes for a power ups effects to stop working	level 2 for 40 seconds, level 3 for 50 seconds, level 4 for 60 seconds and level 5 for 120 seconds.	seconds, level 3 for 50 seconds, level 4 for 60 seconds and level 5 for 120 seconds.	
34	Video of gameplay, where scores are shown at the top right hand corner of the screen, and by inspection, seeing if the score increases as time passes, in a regular pattern.	Score increases by 10 per second.	Score increases by 10 per second.	Fully Successful
35	Video of gameplay where the user clicks on 2 different buttons to lead to 2 different game modes, that are clearly distinguishable	2 separate buttons lead to 2 separate game modes	Separate game modes was not implemented into my game, and so could not be tested	N/A
36	Video of gameplay, where a player collides with a negative power up, and gets the corresponding negative effect for 30 seconds.	Negative effects of power downs are triggered when the player collides with the icon, and last for exactly 30 seconds	Negative power ups were not implemented into my game, and so could not be tested	N/A
37	Video of gameplay of easy mode next to a video of gameplay of hard mode, showing how the speed increase in hard mode is quicker than that of the easy mode.	Escalation of difficulty in easy mode is slower than that of hard mode	Hard mode was not implemented into my game, and so could not be tested	N/A
38	Video of gameplay, and on	Foreground loops without any breaks	Foreground loops without any breaks	Fully Successful

	inspection, the foreground should be looping continuously through the same few options			
39	Video of gameplay, and inspecting the generation of coins to see if they are generated at both random time intervals and at random positions on-screen	The spawning of coins is random	Spawning of coins occurs at random intervals so it is not fully random, however, the position at which they spawn is random, so there is still an aspect of spontaneity	Partially Successful
40	Video of gameplay, where the player collides with a coin in-game and collects them, as shown in a text element on-screen	Names of active power ups shown on-screen	Names of active power ups shown on-screen	Fully Successful
41	Video of abilities menu and of user using coins to upgrade the power ups	The duration of the power up increases	The duration of the power up increases	Fully Successful
42	Video of gameplay, where the user clicks the pause button and is then taken to the pause menu	The entire game stops moving and the player is taken to the pause menu	The first time a user attempts to pause, it pauses the actual gameplay without showing the pause menu UI, but the actual game is paused so the pausing itself does work	Partially Successful
43	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same score as they had before they	The player is on the same score directly after unpausing	The player is on the same score directly after unpausing	Fully Successful

	paused			
44	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same state as they had before they paused	The player is still alive directly after unpausing	The player is still alive directly after unpausing	Fully Successful
45	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same coins as they had before they paused	Same amount of coins collected for the player after the game has been unpause	Same amount of coins collected for the player after the game has been unpause	Fully Successful
46	Video of gameplay, where player pauses the game for a couple seconds, and comes back to have the same power up duration remaining as they had before they paused	Same duration left on the power ups after the game has been unpause	Same duration left on the power ups after the game has been unpause	Fully Successful
47	Video of gameplay where the player leaves the view of the game and them being taken to the game summary menu after	The player dies and is taken to the game summary menu.	The player dies and is taken to the game summary menu.	Fully Successful
48	Video of gameplay where the player collides with an obstacle and then is taken to the game summary menu after	Player dies after colliding with an obstacle	Player dies after colliding with an obstacle	Fully Successful

49	Video of gameplay, and inspecting the generation of individual power ups to see if they are generated at both random time intervals and at random positions on-screen	The generation of power ups in-game is random	The generation of power ups in-game is random	
50	Video of gameplay, where the player dies and is taken to the game summary menu, where accurate statistics from the previous game are shown in on that menu.	All in-game statistics are successfully and accurately transferred from the gameplay to the external database	All in-game statistics are successfully and accurately transferred from the gameplay to the external database	
51	Video of the abilities menu, where the player views the coin balance, then goes into the gameplay, collects coins, where the player dies and is taken to the game summary menu, exits and returns to the Play Menu, then new value of coin balance is checked in the external database to be increased from the original value.	Coin balance would have increased	Coin balance would have increased	
52	Screenshot of external database for a user showing the field that contains their highscore	User data has a field that contains the high score	This was not implemented into my game, and so could not be tested	N/A

53	Screenshot of external database for a user showing the field that contains coin balance	User data has a field that contains the coin balance	User data has a field that contains the coin balance	Fully Successful
54	Screenshot of external database for a user showing the field that contains their selected character	User data has a field that contains the selected character	User data has a field that contains the selected character	Fully Successful
55	Screenshot of external database for a user showing the field that contains their username	User data has a field that contains the username	User data has a field that contains the username	Fully Successful
56	Screenshot of external database for a user showing the field that contains their password	User data has a field that contains the password	User data has a field that contains the password	Fully Successful
57	Screenshot of external database for a user showing the field that contains their email	User data has a field that contains the email	User data has a field that contains the email	Fully Successful
58	Screenshot of the external database showing all of the corresponding fields for all of the user data in one place	User data split by username, with each user having their own database, separate from other users	User data split by username, with each user having their own database, separate from other users	Fully Successful
59	Screenshot of external database for a user showing the field that contains the	User data has a field that contains the power up levels	User data has a field that contains the power up levels	Fully Successful

	power up levels			
60	Screenshot of external database for new user showing that the default power up levels are all 1	New user's power up levels all have the value of 1	New user's power up levels all have the value of 1	Fully Successful
61	Screenshot of external database for new user showing that the default coin balance is 0	New user's coin balance has the value of 0	New user's coin balance has the value of 0	Fully Successful
62	Screenshot of external database for new user showing that the default highscore is 0	New user's high score has the value of 0	This was not implemented into my game, and so could not be tested	N/A

7.1.1 - Further Evidence

This is where all player information is held. Due to PlayFab's security measures, passwords are not able to be viewed, but can be changed on this page:

Master player account ⓘ

Master player account ID



EE43626472CEFF3C

PlayFab username

skibi

PlayFab login email



sigma@gmail.com



Change password



Send password reset email

This is the external database for a pre-existing user who has some game data:

Player Data (Title)

6C55B959B40EF7C4

Players > 6C55B959B40EF7C4 > Player Data (Title)

Overview Cloud Script PlayStream Purchases (Legacy) Statistics Friends Logins Virtual Currency (Legacy) Bans

Data Explorer (basic) Characters Inventory (Legacy) Inventory (V2) Transaction History (V2) **Player Data (Title)** Player Data (Publisher)

Segments Objects Files Policy Groups

Player data

Key *	Value *	Permissions
PowerUpLevels	1,1,1	Private
CoinBalance	2215	Private
SelectedCharacter	FemaleAstronaut	Private
currentCoinsCollected	2	Private
currentScore	80	Private

This is the external database for a new user, with all default values:

Player Data (Title)
EE43626472CEFF3C

Players > EE43626472CEFF3C > Player Data (Title)

Overview Cloud Script PlayStream Purchases (Legacy) Statistics Friends Logins Virtual Currency (Legacy) Bans

Data Explorer (basic) Characters Inventory (Legacy) Inventory (V2) Transaction History (V2) **Player Data (Title)** Player Data (Publisher)

Segments Objects Files Policy Groups

Player data

Key *	Value *	Permissions
SelectedCharacter	MaleAstronaut	Private
CoinBalance	0	Private
PowerUpLevels	1,1,1,1	Private
currentCoinsCollected	0	Private
currentScore	0	Private

...
...

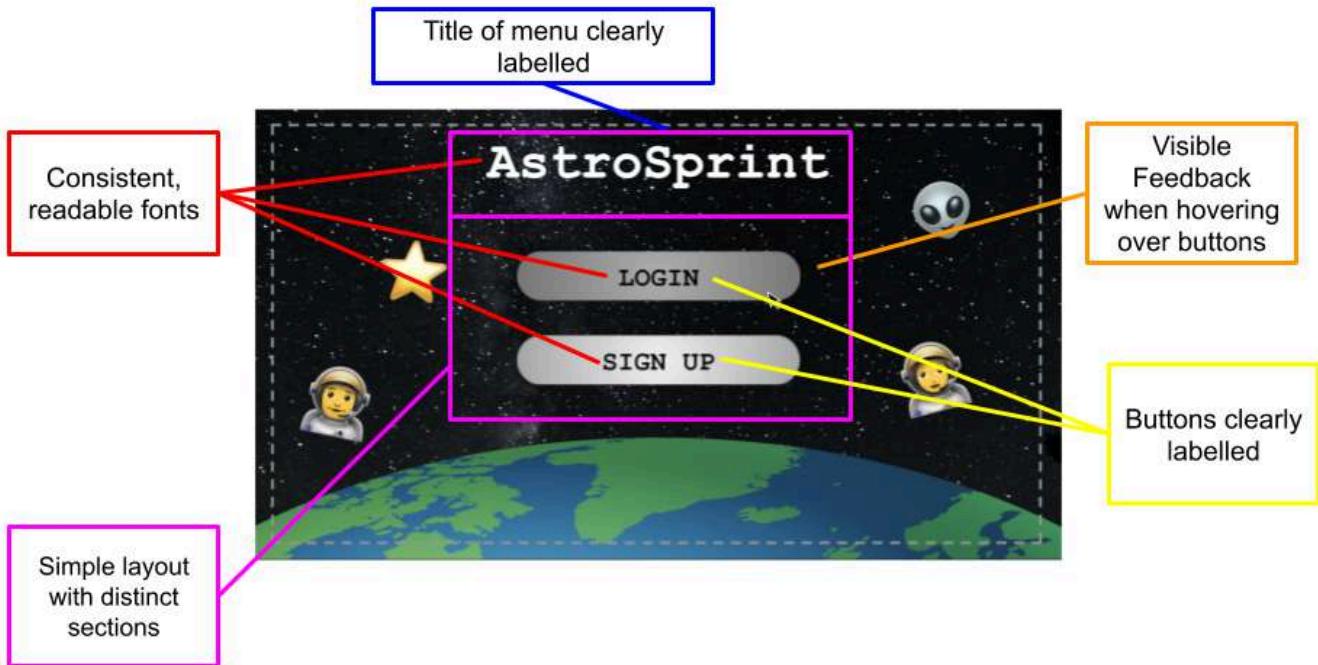
7.2 - Usability Testing

In order to test the usability of my game, I will be using a survey filled out by my primary user, Isabella, to get an opinion on each menu's usability, rating each feature on how usable they are and the game functionality as a whole. I will also be asking how to improve on the usability of each menu and the game as a whole.

At the bottom of the survey, after each section rating how usable each menu is, there's a section for the primary user to give any more features that would improve the system's usability.

7.2.1 - Main Menu Usability

7.2.1.1 - Evidence of Features



7.2.1.2 - Primary User Feedback

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Buttons clearly labelled	All buttons should be clearly labelled, easy to read, and reflect the menu that they lead to	10	Success	Buttons are clearly labelled
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	9	Success	On the sides of the buttons, there's empty space where if you hover over, the button gives visual feedback even though the mouse isn't on the button [Evidence] 

Simple layout with distinct sections	Main menu should have a simple layout, as there are only 2 options within it	6	Partial Success	Layout is good, but buttons could be a bit bigger so they are the focus of the menu. With the characters and earth image at the bottom, the buttons don't stand out much
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	Font is same on entire menu and is readable
Centralised positioning of buttons	Buttons should be positioned in the centre of the screen so they are easy to find	10	Success	Buttons are in the centre of the menu
Title of Menu Clearly Labelled	Menu should have a title at the top so the user knows what menu they are looking at	7	Partial Success	Although it is a main menu, so it is reasonable to title this menu the name of the game, this title does not describe the menu in any way.

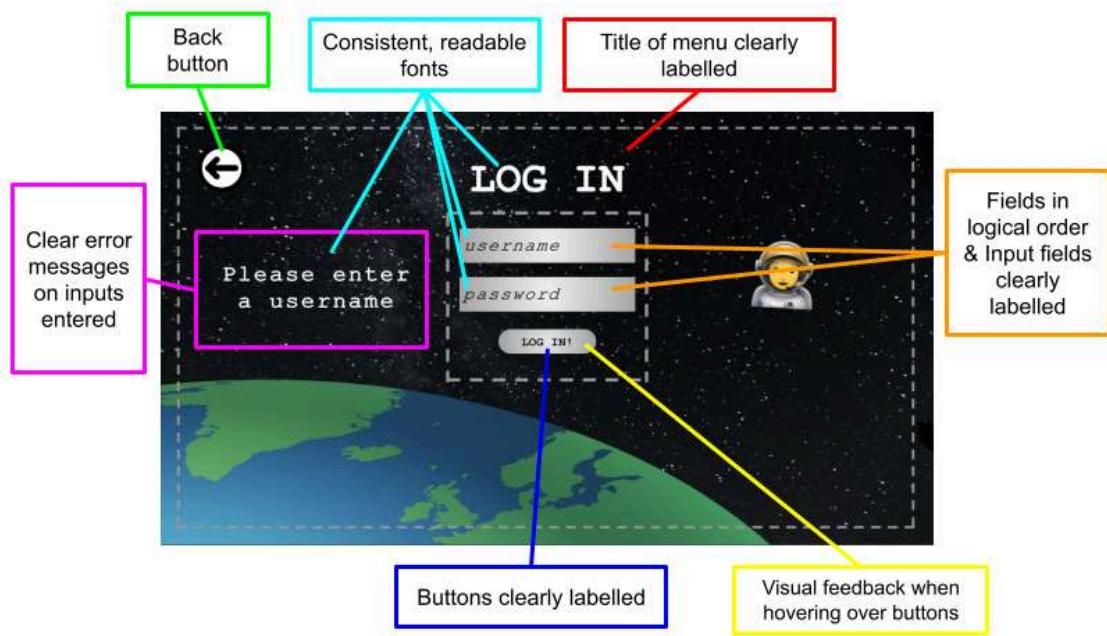
7.2.1.3 - Suggested Features

Suggested Features to Add (From Primary User) :

- An option to exit the game from this menu, as there currently isn't one
- Audio feedback for buttons, as they don't currently have any and the game is silent

7.2.2 - Login Menu Usability

7.2.2.1 - Evidence of Features



7.2.2.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Input fields clearly labelled	Input fields are clearly labelled so it is easy to understand	10	Success	Input fields are clearly labelled with what information should be inputted
Clear error messages on inputs entered	Necessary so that user knows what any issues are when trying to log in	10	Success	Appropriate, readable error messages for any issue when logging in
Buttons clearly labelled	Buttons being clearly labelled enables users to understand what they are clicking on, and summarise	2	Failure	Login button is small and hard to read because the text is so small

	the function of the button			
Back button	Necessary for smooth navigation between menus, and so that users can return to main menu if needed	10	Success	Back button is clearly in top left of the screen
Fields in logical order	Fields are in the order of username, then password, because the validity of the password is to gain access to a specific user, not the other way around	10	Success	Field order makes sense
Consistent, readable fonts	Fonts throughout the menu should be the same, and easy to read	6	Partial Success	Text on login button itself is difficult to read, but the rest of the text is easy to read and the rest of the menus are
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	The label at the top of the screen says 'Log In' and it accurately describes what the menu is for
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	1	Failure	There is visible feedback when clicking over the 'login' button, but none when hovering over the button

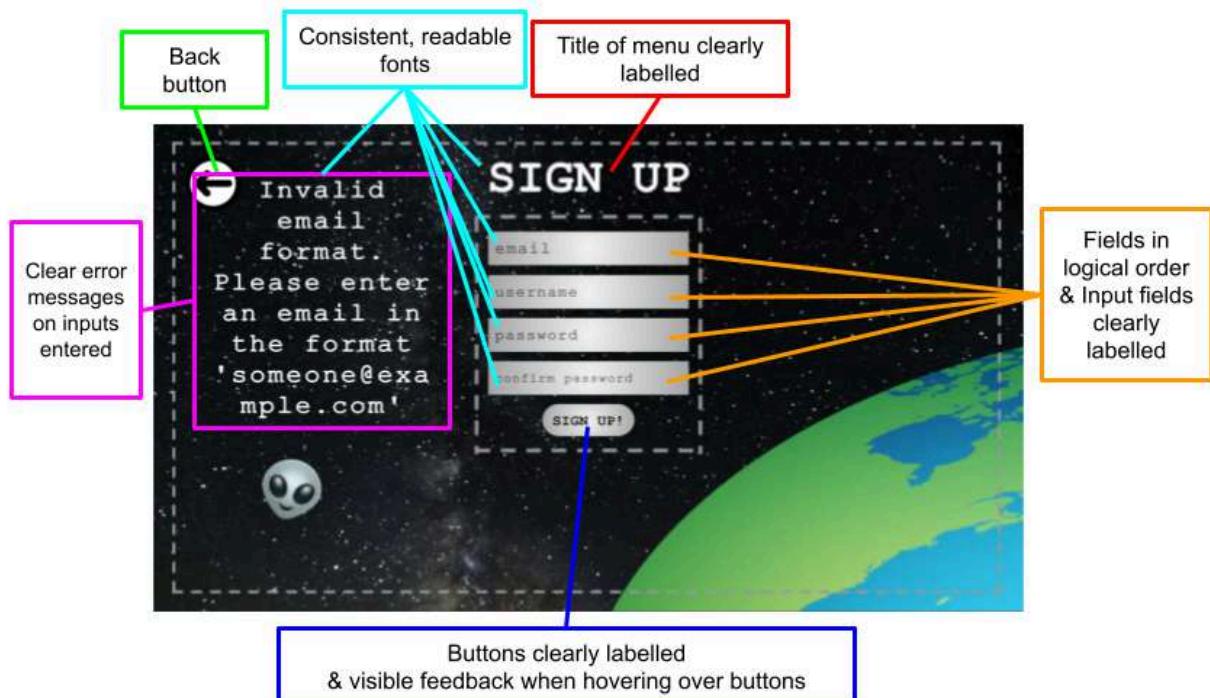
7.2.2.3 - Suggested Features

Suggested features for improved usability from primary user:

- Make it so that the password isn't visible when typing, it should be censored for security, with an option to hide or show

7.2.3 - Sign Up Menu Usability

7.2.3.1 - Evidence of Features



7.2.3.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Input fields clearly labelled	Input fields are clearly labelled so it is easy to understand	10	Success	Input fields are labelled clearly and I can understand what to input in each
Clear error messages on inputs entered	Necessary so that user knows what any issues are when trying to log in	7	Partial Success	Appropriate, readable error messages for any issue when signing up for all error messages, apart from the email formatting one, which the words can't all fit on one line so it looks jumbled
Buttons	Buttons being	3	Partial	Actual sign up button

clearly labelled	clearly labelled enables users to understand what they are clicking on, and summarise the function of the button		Success	could be bigger so it's easier to read
Back button	Necessary for smooth navigation between menus, and so that users can return to main menu if needed	10	Success	Back button is present and visible and takes user back to main menu
Fields in logical order	Fields are in the order of email, username, then password, then confirm passwords because the email and username need to be unique, and the confirm password has to confirm the previously entered password, so has to be above it	10	Success	Fields are in a logical order
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	Title accurately describes what the menu is
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	5	Partial Success	There is very slight change in colour when hovering, but it is not obvious
Consistent, readable fonts	Fonts throughout the	7	Partial Success	Text on actual sign up button itself is

	menu should be the same, and easy to read			somewhat hard to read, but the rest of the text is easy to read and the rest of the menus are
--	---	--	--	---

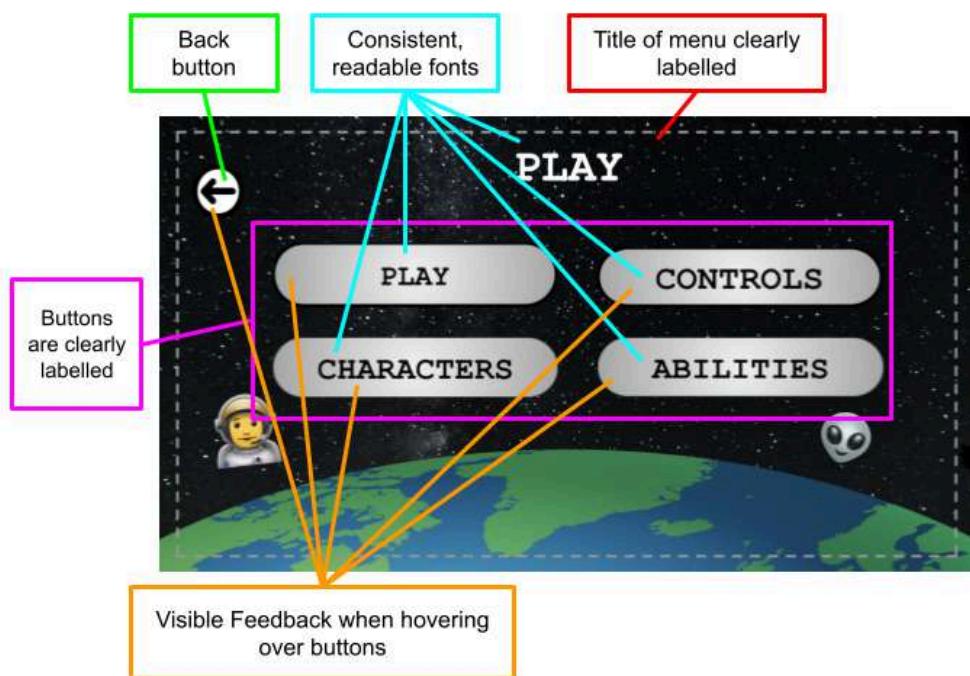
7.2.3.3 - Suggested Features

Suggested features for improved usability from primary user:

- Same as login menu: Make it so that the password and confirm password fields aren't visible when typing, it should be censored for security, with an option to hide or show
- Have an information panel somewhere showing what the formats and types of characters are allowed to be in each field, as it's not clear to user before inputting an invalid input

7.2.4 - Play Menu Usability

7.2.4.1 - Evidence of Features



7.2.4.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User

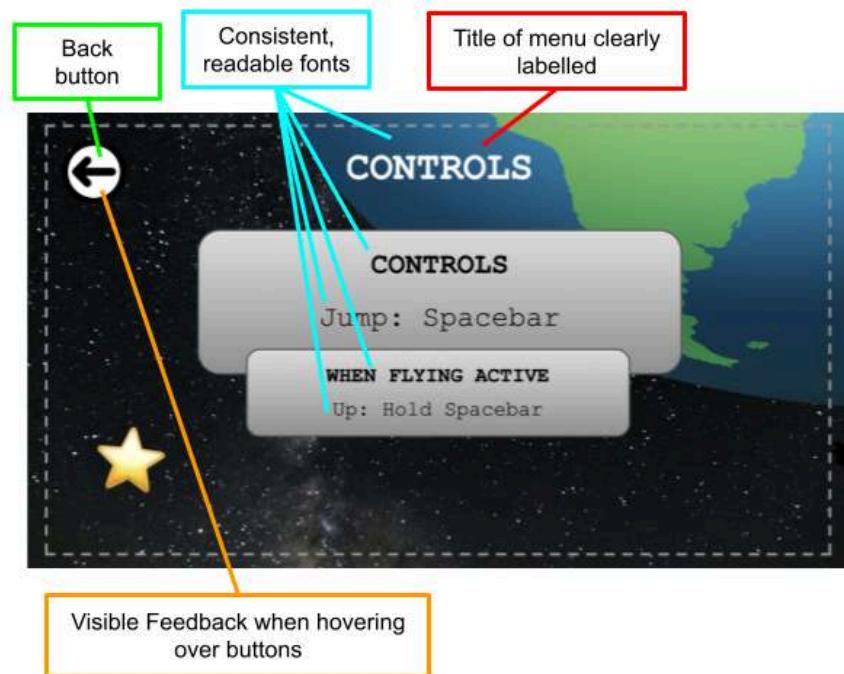
Buttons clearly labelled	Necessary for smooth navigation between menus, and so that users can return to main menu if needed	8	Success	The play button is confusing when the menu is also called play, but the rest of the labels are fine
Back button	Necessary for smooth navigation between menus, and so that users can return to main menu if needed	10	Success	Play menu back button works and takes player back to main menu
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	3	Failure	Slightly confusing because the menu says play but there is also a button that is also called play, and you can't play anything on the menu that says play at the top
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	All fonts are consistent and readable across the menu
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	0	Failure	There is no visible feedback when hovering over buttons

7.2.4.3 - Suggested Features

No features to add, just changing the title of the menu so it more clearly reflects the contents of it

7.2.5 - Controls Screen Usability

7.2.5.1 - Evidence of Features



7.2.5.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Back button	Necessary for smooth navigation between menus, and so that users can return to play menu if needed	10	Success	Back button on the controls menu takes you back to the play menu
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	The title accurately reflects the contents of the menu
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	All fonts are consistent and readable across the menu
Visible feedback when	All buttons should become darker when	4	Partial Success	There is visible feedback when hovering

hovering over buttons	hovered over, to indicate that button is being pressed			over the back button, but it's extremely minimal, not obvious at all
-----------------------	--	--	--	--

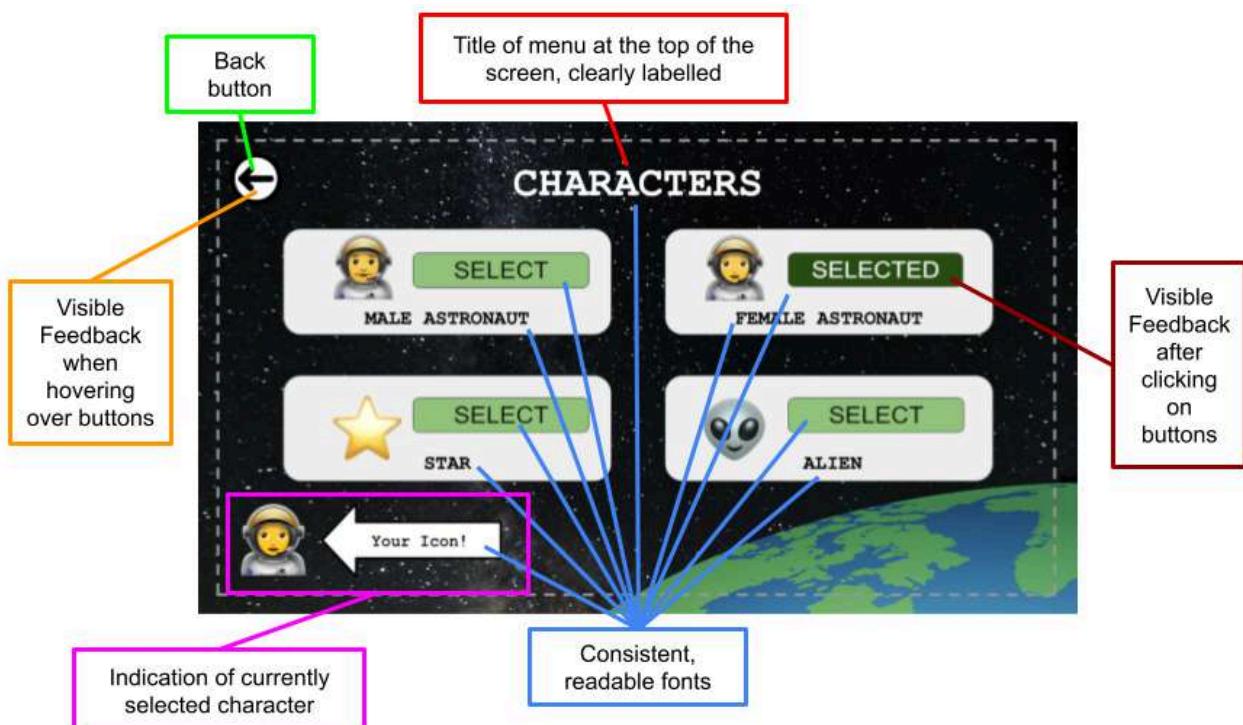
7.2.5.3 - Suggested Features

Suggested features to make menu more usable from primary user

- Menu should be like an actual tutorial instead, where it shows you all of the information you need to play the game (no. of lives, description of all power ups, etc.) and guides you through using all of them with written instructions

7.2.6 - Character Menu Usability

7.2.6.1 - Evidence of Features



7.2.6.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Back button	Necessary for smooth navigation between	10	Success	Back button on the character menu takes

	menus, and so that users can return to play menu if needed			you back to the play menu
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	The title accurately reflects the contents of the menu, the characters selection menu
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	5	Partial success	All fonts are readable on the whole menu, but the fonts on the select/selected buttons is different to the rest of the menu
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	2	Failure	There is visible feedback when hovering over the back button, but not for any of the character selection buttons
Visible feedback after clicking on a button	All character selection buttons should go from 'Select' to 'Selected' to indicate to the user that they have selected the button	10	Success	Light green 'Select' button turns to a dark green 'Selected' button after it has been pressed
Indication of currently selected character	The user should be able to see their currently selected character separately from the catalogue of characters, so it is explicitly clear what characters have been selected	10	Success	There is an arrow at the bottom that points to a separate image that shows the currently selected character
Default character selected	There should be a character selected for the player by default, even if they don't select a character	10	Success	The male astronaut character is selected automatically

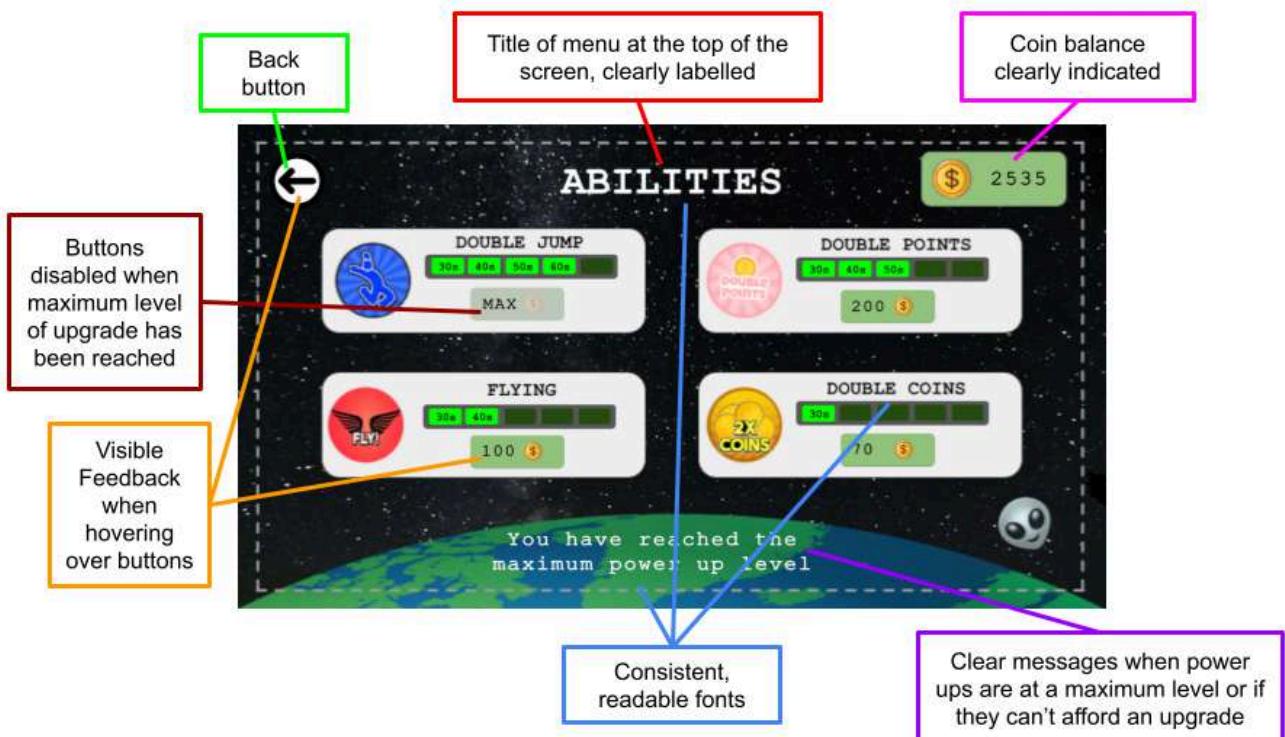
7.2.6.3 - Suggested Features

Suggested from primary user

- Change the font on the 'select' and the 'selected' buttons to be the same as the rest of the menu

7.2.7 - Abilities Menu Usability

7.2.7.1 - Evidence of Features



7.2.7.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Back button	Necessary for smooth navigation between menus, and so that users can return to play menu if needed	10	Success	Back button on the character menu takes you back to the play menu
Title of menu at top of	Menu should have a title at the top so	10	Success	The title accurately reflects the contents

screen	the user knows what menu they are looking at			of the menu, the characters selection menu
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	All fonts are both consistent and readable on the whole menu
Coin balance clearly indicated	It should be easy for the user to see the numbers that show the coin balance, and it should be obvious that the value on that part of the screen is the coin balance	9	Success	It's very clear that the coin balance is the thing in the top right corner, but the only confusing this is that the icon for the coins is different in the abilities menu compared to the actual gameplay
Buttons disabled when maximum level of upgrade has been reached	Upgrade buttons should not be able to be clicked after a maximum upgrade has been reached, so the user knows that they cannot upgrade the power up anymore and they don't accidentally waste their coins on a power up that won't upgrade further.	5	Partial Success	It does disable the button that is at the maximum upgrade, but it makes the other buttons also not work unless you click once or twice
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	2	Failure	There is visible feedback when hovering over the back button, but not for any of the price upgrade buttons
Clear messages when power ups are at maximum level or if they can't afford an upgrade	Necessary so that user knows why they can't make certain purchases	10	Success	Appropriate, readable error messages for any issue when attempting to upgrade

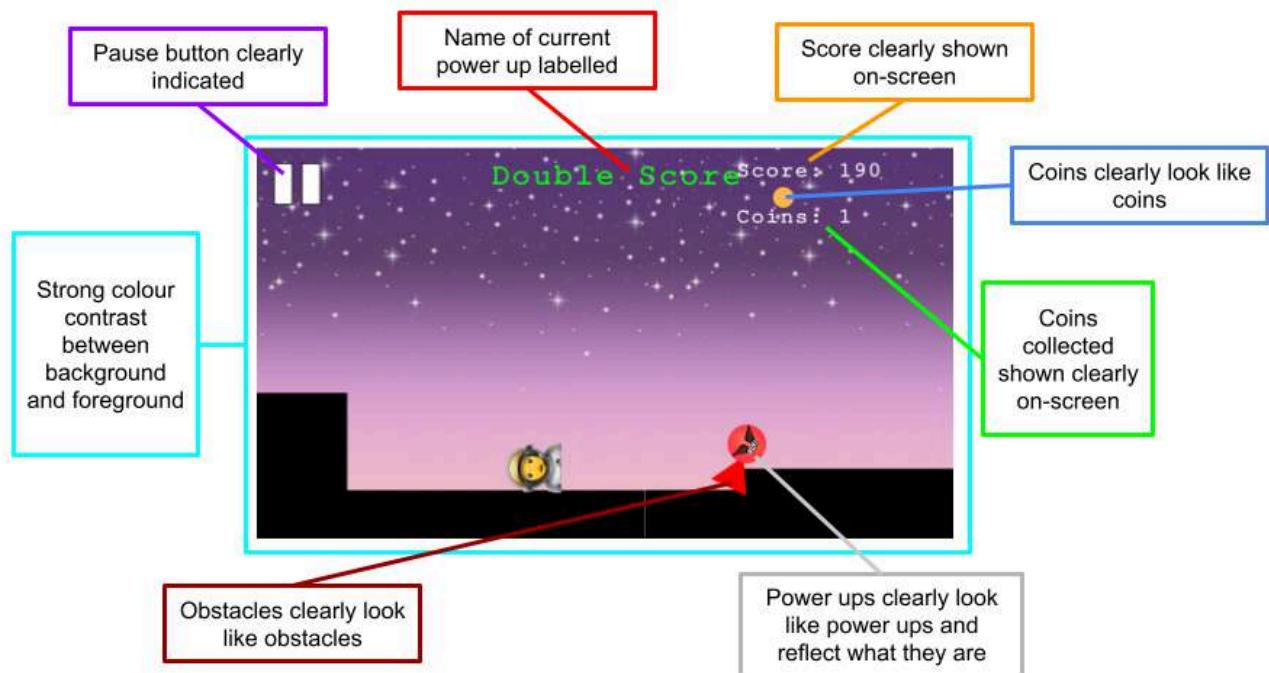
7.2.7.3 - Suggested Features

Suggested features

- Make it more clear what the upgrades are for, possibly by giving a short description on what the upgrades are
- Make the coin images in-game be consistent with the ones in the gameplay
- Decrease (or ideally completely remove) time delay in displaying the coin balance in Unity
- There's sometimes issues in trying to click upgrade buttons, so you have to go off the menu and back on to do them, this needs to be fixed

7.2.8 - Gameplay Usability

7.2.8.1 - Evidence of Features



7.2.8.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Name of current power up labelled	Name of power up is shown on screen so that the user knows	5	Partial Success	It does show what power up is active, but if you collect 2 power ups, it doesn't say the

	what they are			name of the first one, only the second one
Score clearly shown on-screen	So that the user can clearly, and therefore quickly, see what their current score is, without being distracted from the game	8	Success	Can see the score, but it sort of blends in with the white stars from the background, so is slightly harder to read, but still doable
Coins collected shown clearly on-screen	So that the user can clearly, and therefore quickly, see what their current coins collected is, without being distracted from the game	8	Success	Can see the coins collected, but it sort of blends in with the white stars from the background, so is slightly harder to read, but still doable
Pause button clearly indicated	The pause button should be clearly indicated so the user can quickly access it	10	Success	Pause button is large and obvious
Strong colour contrast between background and foreground	There should be a large contrast of colour between them so that it is clear to the user where the foreground is and where the background is	9	Success	There is a very bold contrast between the background and foreground
Obstacles clearly look like obstacles	The obstacles should be very obviously obstacles so the player knows to avoid them	8	Success	The obstacles look like obstacles, as they look red and pointy
Coins clearly look like	The coins should be very	8	Success	The coins looks like coins, because they

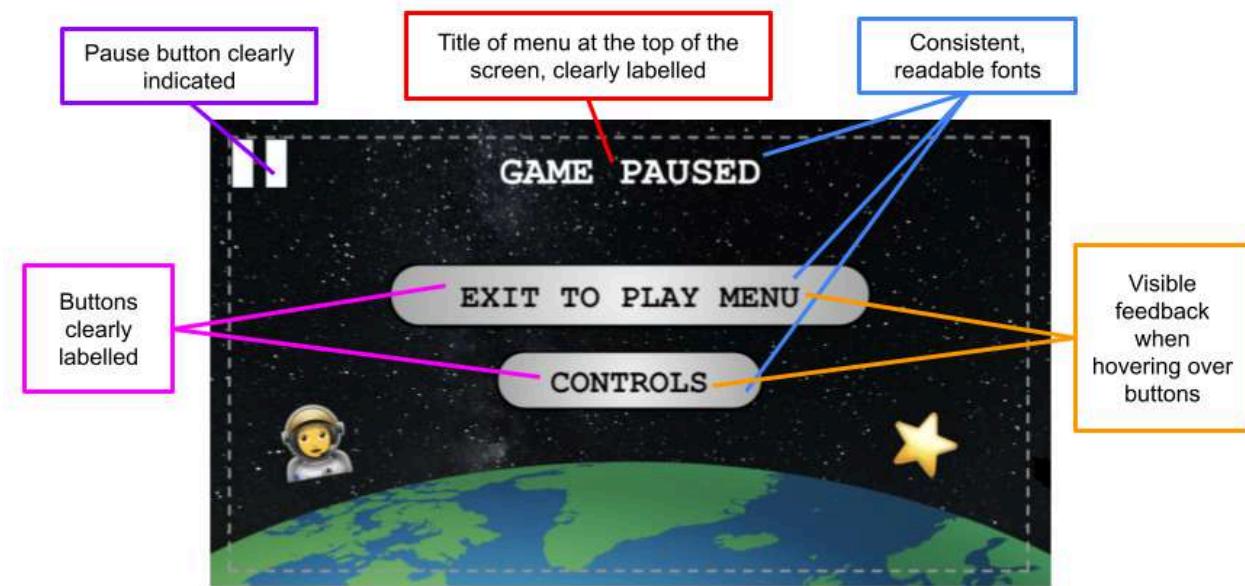
coins	obviously coins so the player knows to collect them			
Power up icons look like power ups, and icons clearly reflect what they are	The power up icons should accurately reflect the power up they represent and should look obviously like power ups so the user knows to collect them	7	Partial Success	<p>The double coins and double coins look like power up icons, but the flying icon seems to look like an obstacle at first glance, because it is also red.</p> <p>The double jump icon doesn't look like that much of an accurate representation of double jump, just jumping general</p>

7.2.8.3 - Suggested Features

- Make the coin images in-game be consistent with the ones on the abilities menu
- Have information written on the background (that eventually disappears with the game movement) that describes what each of the objects mean (yellow circle = coins, red triangle = obstacles, etc.)
- Have information about number of lives
- The remaining duration of a power up should be shown on the screen
- When the 'Double Score' label is on screen, it should not overlap with the score text

7.2.9 - Pause Menu Usability

7.2.9.1 - Evidence of Features



7.2.9.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	The title accurately reflects the contents of the menu
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	All fonts are both consistent and readable on the whole menu
Pause button clearly indicated	The pause button should be clearly indicated so the user can quickly access it	10	Success	Pause button is large and obvious
Buttons	Buttons should be	9	Success	Buttons are extremely

clearly labelled	clearly labelled with their function so the user can understand their function			clear on what they are for, and are readable.
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	2	Failure	There is a slight visible feedback (button becomes slightly darker) but not very noticeable

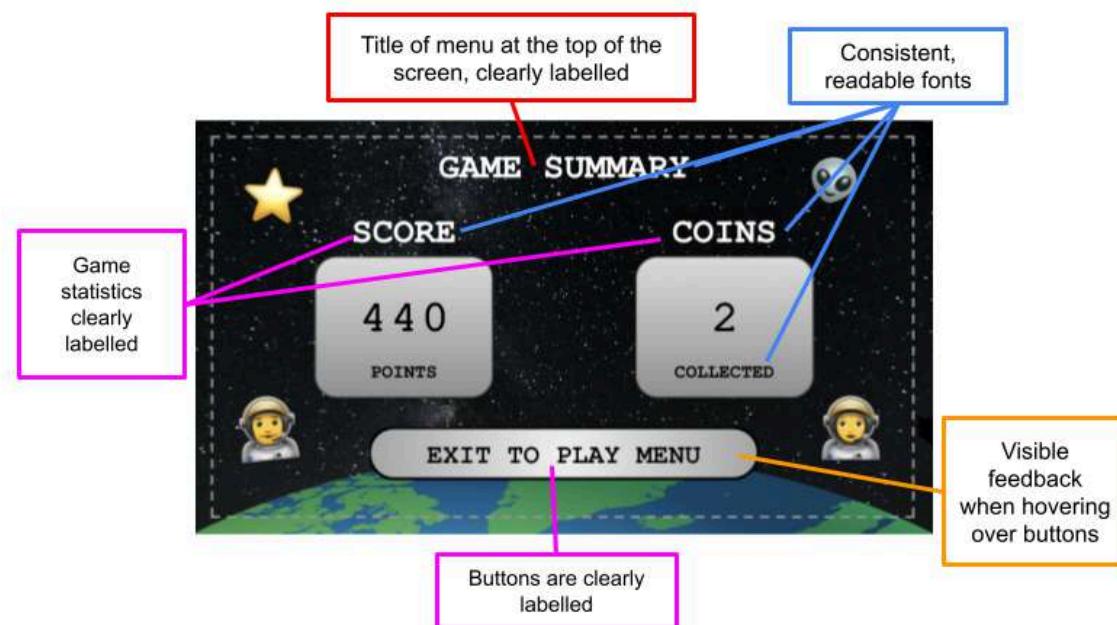
7.2.9.3 - Suggested Features

Suggested usability features from primary user

- Ask the user if they are sure they want to leave when they click 'Exit To Play Menu' before actually taking them to the Play Menu
- Tell the user that their information for this previous game info (score and coins) will not be saved if they leave the game through the pause menu
- Change 'Controls' to 'View Controls' so it is more clear

7.2.10 - Game Summary Menu Usability

7.2.10.1 - Evidence of Features



7.2.10.2 - Feedback from Primary User

Feature to Test	Expected Usability	Rating from Primary User /10	Success, Partial Success or Failure as Effective Usability Feature	Comments From Primary User
Title of menu at top of screen	Menu should have a title at the top so the user knows what menu they are looking at	10	Success	The title accurately reflects the contents of the menu, the game summary menu
Consistent, readable fonts	All text should have the same font, so that the game remains cohesive. Font should be large & readable so users can understand them	10	Success	All fonts are both consistent and readable on the whole menu
Buttons clearly labelled	Buttons should be clearly labelled with their function so the user can understand their function	10	Success	Buttons are extremely clear on what they are for, and are readable.
Visible feedback when hovering over buttons	All buttons should become darker when hovered over, to indicate that button is being pressed	0	Failure	There is no visible feedback for any of the buttons.
Game statistics clearly labelled	The values for the statistics of the most recent game should be labelled so it is clear to the user what they represent (number of coins or score)	10	Success	It is very obviously labelled what each value means.

7.2.10.3 - Suggested Features

- Have an option to play again straight from the game summary menu, as it would make playing the game again much more convenient, as you don't have to navigate back to the play menu and then click play again to be able to play again
- Get rid of time delay for showing the final score and final coins

7.3 - Fulfillment of Success Criteria

All references to videos in this section refer to the video of the final iteration, named '3002_RichelleAmaAcheampong_H446-03_FinalIteration'.

7.3.1 - Menus & User Interface

Success Criteria	Criteria No.	Fully Met, Partially Met or Not Met?	Evidence (Test no. &/or Explanation)
All menus titled	1	Fully met	<p>Test no. 1</p> <p>As shown in the final iteration video, there are titles at the top of every single menu, with the only part of the game not having a title being the actual gameplay itself</p>
Clearly labelled menus, with options within them being clear	2	Partially met	<p>Test no. 2</p> <p>As shown in the final iteration video, every single menu has been labelled with what it is called. These are all clear, apart from the Play Menu label, which just says 'Play'. The title 'Play' is not descriptive of the options within the play menu, and is somewhat confusing because there is an option on this menu that also says 'play'. Therefore, this criteria has been partially met.</p>
Main Menu UI	3	Fully met	<p>Test no. 3</p> <p>In my final iteration video, the user is able to click to view and interact with the main menu, click all of its buttons to go to the login or sign up menus. This means that the user interface for the main menu is fully functional. Therefore, this success criteria has been fully met.</p>
Login Menu UI	4	Fully met	<p>Test no. 4</p> <p>In my final iteration video, the user is able to click to view and interact with the login menu, click all of its buttons to go to log in or go back to the main menu menus, and input information into its input fields. This means that the user interface for the login menu</p>

			is fully functional. Therefore, this success criteria has been fully met.
Sign Up Menu UI	5	Fully met	<p>Test no. 5</p> <p>In my final iteration video, the user is able to click to view and interact with the login menu, click all of its buttons to go to sign up or go back to the main menu menus, and input information into its input fields. This means that the user interface for the sign up menu is fully functional. Therefore, this success criteria has been fully met.</p>
Play Menu UI	6	Fully met	<p>Test no. 6</p> <p>In my final iteration video, the user is able to click to view and interact with the character menu, click all of its buttons to go to the gameplay, controls screen, character menu or abilities menu. This means that the user interface for the player menu is fully functional. Therefore, this success criteria has been fully met.</p>
Tutorial Screen UI	7	Partially met	<p>Test no. 7</p> <p>The tutorial screen, in later iterations after the success criteria was written, was re-named to be a controls menu to more accurately reflect its contents. Due to the fact that the interface does not fit the description of a 'Tutorial', to the best it can be. It does give the user how to use the controls, but doesn't actually guide the player through real gameplay. Therefore, this criteria has been partially met.</p>
Character Menu UI	8	Fully met	<p>Test no. 8</p> <p>In my final iteration video, the user is able to click to view and interact with the character menu, click all of its buttons to select characters or go back to the play menu. This means that the user interface for the character menu is</p>

			fully functional. Therefore, this success criteria has been fully met.
Abilities Menu UI	9	Fully met	<p>Test no. 9</p> <p>In my final iteration video, the user is able to click to view and interact with the abilities menu, click all of its buttons to upgrade or go back to the play menu. This means that the user interface for the abilities menu is fully functional. Therefore, this success criteria has been fully met.</p>
Pause Menu UI	10	Fully met	<p>Test no. 10</p> <p>In my final iteration video, the user is able to click the pause button in-game to view and interact with the pause menu, click all of its buttons to view the controls, go back to the gameplay or exit to the play menu. This means that the user interface for the pause menu is fully functional. Therefore, this success criteria has been fully met.</p>
Game Summary UI	11	Fully met	<p>Test no. 11</p> <p>In my final iteration video, the user is able to die and then view and interact with the pause menu, click its button to exit to the play menu, and view their statistics. This means that the user interface for the game summary menu is fully functional. Therefore, this success criteria has been fully met.</p>
Back buttons on all menus	12	Partially met	<p>Test no. 12</p> <p>In the final iteration video, there is a back button on every single menu (Play, abilities, characters, controls, login, sign up, pause, game summary) apart from the main menu. Due to this, there is a back button on the majority of the menus, therefore, not every single menu has a back button. Therefore, this success criteria was partially met.</p>

Non-bright, cohesive colour scheme	13	Fully met	<p>Test no. 13</p> <p>As shown in my final iteration video, there are no colours within AstroSprint that are extremely bright and don't fit in the colour scheme. The colour palette of my final interaction closely matches that of the one in my design, which was previously decided to be cohesive. This means that my colour scheme is cohesive, and therefore, this criteria has been met.</p>
Varied environments, with users being able to play in places with different backgrounds	14	Not met	<p>Test no. 14</p> <p>In the final iteration video, when the user clicks on 'Play' there is 1 consistent background throughout the entire gameplay. This is the same background every single time a game ends and restarts. This means there is only 1 background, and therefore only 1 environment. Therefore, this criteria has not been met.</p>
Login system	15	Fully met	<p>Test no. 15</p> <p>In the final iteration video, the user is able to go onto the login menu, be told an error message next to the inputs when they entered invalid inputs, and when correct credentials were entered, were taken to the 'Play' Menu. These are all clear indicators that the login system is fully functional. Therefore, this success criteria has been met.</p>
Sign up system	16	Fully met	<p>Test no. 16</p> <p>In the final iteration video, the user is able to go onto the sign up menu, be told an error message next to the inputs when they entered invalid inputs, and when valid credentials were entered, were taken to the 'Play' Menu. These are all clear indicators that the sign up system is fully functional. Therefore, this success criteria has been met.</p>

Catalogue of characters that the player can choose from	17	Fully met	<p>Test no. 17</p> <p>In the final iteration video, when the user clicks on the 'Characters' button in the play menu, they are taken to the character menu, in which there are 4 different options of characters for the user to select from. As there are multiple options shown to the user that they can select from, this means there is a catalogue of characters that the player can choose from. Therefore, this success criteria has been fulfilled.</p>
Input validation for email when signing up	18	Fully met	<p>Test no. 18</p> <p>In the final iteration video, the user attempts to leave the email field blank, which triggers an error message and doesn't allow them to sign up. There is then another attempt to input an email in an invalid format, which is also met with an error message and the user is not able to sign up. The error messages are the result of there being a presence check and a format check, which are methods of input validation. Therefore, there is input validation for the email field when signing up. Therefore, this success criteria has been fully met.</p>
Input validation for username when signing up	19	Fully met	<p>Test no. 19</p> <p>In the final iteration video, the user attempts to leave the username field blank, which triggers an error message and doesn't allow them to sign up. There is then another attempt to input a username that contains special characters, which is also met with an error message and the user is not able to sign up. The error messages are the result of there being a presence check and a format check, which are methods of input validation. Therefore, there is input validation for the username field when signing up. Therefore, this success criteria has been fully met.</p>

Input validation for password when signing up	20	Fully met	<p>Test no. 20</p> <p>In the final iteration video, the user attempts to leave the password field blank, which triggers an error message and doesn't allow them to sign up. There is then another attempt to input a password that is shorter than 8 characters, which is also met with an error message and the user is not able to sign up. The error messages are the result of there being a presence check and a length check, which are methods of input validation. Therefore, there is input validation for the password field when signing up. Therefore, this success criteria has been fully met.</p>
Comparison between password and confirm password fields	21	Fully met	<p>Test no. 21</p> <p>In my final iteration video, there is an attempt to type in 2 different inputs into the password and confirm password fields. This is met with an error message, which alerts the user to ensure that the password and confirm password fields match. This error message could only pop up for 2 differing inputs if there was a comparison between the password and confirm password fields. Therefore, there is a comparison between the password and confirm password fields. Therefore, this success criteria has been fully met.</p>
Coins balance retrieved from external database in abilities menu	22	Fully met	<p>Test no. 22</p> <p>In the final iteration video, as there was a login for a pre-existing user, and when they entered the abilities menu, their current coin balance loaded. There is also a field within my external database that holds the current coin balance of the user. Therefore, this success criteria was met.</p> <p>The field from PlayFab is shown here:</p>

			<table border="1"> <thead> <tr> <th>Key *</th><th>Value *</th></tr> </thead> <tbody> <tr> <td>CoinBalance</td><td>221</td></tr> </tbody> </table>	Key *	Value *	CoinBalance	221
Key *	Value *						
CoinBalance	221						
Coin balance updated in external database after every upgrade purchase	23	Fully met	<p>Test no. 23</p> <p>In my final iteration video, there is a purchase made of power up upgrades for a pre-existing user, in which they start at a balance of 3695, retrieved from the external database.</p> <p>After making all of their purchases, their balance is shown on-screen to be 2215. After checking the external database, this value was also held here:</p> <table border="1"> <thead> <tr> <th>Key *</th><th>Value *</th></tr> </thead> <tbody> <tr> <td>CoinBalance</td><td>2215</td></tr> </tbody> </table> <p>This proves that the coin balance is updated in the external database after every upgrade purchase. Therefore, this criteria has been met.</p>	Key *	Value *	CoinBalance	2215
Key *	Value *						
CoinBalance	2215						
Indication of which character was chosen in Character Menu	24	Fully met	<p>Test no. 24</p> <p>In the final iteration video, the user clicks on the 'Select' buttons for all of the different characters, which changes them from 'Select' to 'Selected'. The selection of each character also caused the image in the bottom right corner (next to the arrow that says 'Your Icon!') to change to whatever power up is selected. The changing of this button's design and the changing of the icon next to the 'Your Icon' button is clearly indicating what characters have been chosen in the character menu. Therefore, this success criteria has been fully met.</p>				
Indicator of each power up's level	25	Fully met	<p>Test no. 25</p> <p>In the final iteration video, the power up level is shown through the</p>				

			green bars next to the power up icon. They increase once the user (with enough coins) clicks on them, indicating how upgraded the power up is. Therefore, this success criteria has been met.
There is a default selected character	26	Fully met	<p>Test no. 26</p> <p>In my final iteration video, before selecting any character, the default character for a new user that has just signed up is the male astronaut. This is definitely a default character because there are no buttons on the character menu that say 'selected', they all say 'select'. This clearly indicates that there are no options currently selected. If there are no options currently selected, and there is still a character selected, this must be a default option. Therefore, there is a default selected character. Therefore, this success criteria has been fully met.</p>

7.3.2 - Gameplay

Success Criteria	Criteria No.	Fully Met, Partially Met or Not Met?	Evidence (Test no. &/or Explanation)
Implementing challenges to gain in-game currency	27	Not met	<p>This feature was not present in my game. As seen in the final iteration video, there is no option throughout the game to see any challenges, and there is no menu in which challenges are shown or mentioned. As there is no mention of the feature in this criteria, it has not been met.</p>
Game being 2D	28	Fully met	<p>Test No. 28</p> <p>This is evident throughout the entire final iteration video. The entire game is 2D, and this is obvious by inspection. Additionally, in all of my code, no z-axis vectors are used. There are no 3D aspects of my game, therefore</p>

			this point is fully met.
Slow escalation in difficulty in the easier mode of the game	29	Fully met	<p>Test No. 29</p> <p>As shown in the final iteration video, after a long period of gameplay, the movement of the foreground becomes increasingly fast. This takes a considerable period of time to get somewhat difficult (after around 100 seconds in the final iteration video), therefore, this is a slow escalation in difficulty. Therefore, this success criteria was fully met.</p>
Indication of what power ups are active to user	30	Partially met	<p>Test No. 30</p> <p>As shown in my final iteration video, in the gameplay, there is green text at the top of the screen that says the name of the currently active power up, from when it is activated, to when it runs out. However, when a second, different, power up is also collected by the player, the text changes to the name of the second power up.</p> <p>Because the game does not show the name of all currently active power ups, only the most recently activated one, this criteria is only partially met.</p>
Indication of power up remaining duration when active	31	Not met	This feature was not present in my game. As seen in the final iteration video, there is no indication or reference to the remaining duration of a power up after it has been collected. As there is no mention of the feature in this criteria, it has not been met.
Power-ups	32	Fully met	<p>Test No. 32</p> <p>As shown in my final iteration video, there are power ups in my game. These, in the video, are seen being used (being able to jump twice when double jump is active, being able to continually ascend when flying is active, and going 20 points every second instead of 10</p>

			when double score is active) and therefore, this success criteria has been met.
Power up duration according to level upgraded	33	Fully met	<p>Test No. 33</p> <p>In my final iteration video, all power ups are upgraded to level 4, which should mean the duration of them is 60s. In my gameplay, after collecting a power up, the effect lasted for 60 seconds for all of them. This means that the power duration is based on the level, because if it wasn't it would default to the original 30 second time period for power ups, but it didn't. Therefore, the success criteria was met.</p>
Scoring system based on how long you last in-game	34	Fully met	<p>Test No. 34</p> <p>During the video of the final iteration gameplay, the score is shown in the corner to increase by 10 every 1 second. As the score increases by 10 per second, the score is based upon how many seconds a player lasts within the game. This means that the scoring system is based on how long you last in-game. Therefore, this success criteria has been fully met.</p>
Different game modes	35	Not met	<p>In the final iteration video, when the user is on the play menu and all the options are showing, there is only 1 option to be able to engage in actual gameplay. There is only 1 game mode, as every single button in the game was pressed and only 1 of them lead to any sort of gameplay. Therefore, there is only 1 game mode in my game. Therefore, there is only 1 game mode in my game, which is due to a lack of time. Therefore, this success criteria has not been met.</p>
Implementation of negative power ups in a harder mode	36	Not met	<p>Due to the fact that I did not have enough time to implement hard mode, I was not able to implement any negative power ups in a harder mode. There is only 1 game mode in my game. Therefore, this success</p>

			criteria was not met.
Implementation of a sharp escalation in difficulty in the harder mode, in comparison to the easy mode	37	Not met	Due to the fact that I did not have enough time to implement hard mode, I was not able to implement a sharp escalation in difficulty in a harder mode. There is only 1 game mode in my game. Therefore, this success criteria was not met.
Game is endless	38	Fully met	<p>Test No. 38</p> <p>In the video of my final iteration's gameplay, the only thing that ended a game was the player dying through leaving the view or colliding with an obstacle. Due to this and the fact that the foreground is 4 game objects in a loop, that reset back to directly outside the camera repeatedly, there is no stopping condition for the game apart from player death and intentionally quitting through the pause menu. There is no indication, nor is there any code, that proves that the game will end after a certain period of time or after a certain distance is reached. Therefore, it can be concluded that the game goes on until the player dies or intentionally exits to the play menu through the pause menu. Therefore, this success criteria has been fully met.</p>
Random generation of in-game currency within the playing environment	39	Partially Met	<p>Test No. 39</p> <p>In the final iteration video, in the gameplay, it is clear on inspection that in-game currency (coins) are being spawned from the top of the screen in AstroSprint. They spawn from random positions on the x-axis across the screen, so they spawn at different points across the screen. However, the coins spawn at regular time intervals, so the spawning of them is not completely random, only partially. Therefore, the generation of in-game currency within the playing environment is only partially random. Therefore,</p>

			this success criteria has only been partially met.
Collection of in-game currency after colliding with it	40	Fully met	<p>Test No. 40</p> <p>In the final iteration video, the user enters gameplay, in which they collide with a coin and text appears on the top right corner of the screen that says 'Coins: 1' and the coin disappears. These are both clear indicators that the player has collected a coin after colliding with it. Therefore, this success criteria has been fully met.</p>
Ability to upgrade power-ups outside of the playing environment	41	Fully met	<p>Test No. 41</p> <p>In my final iteration video, the user was able to go into the abilities menu and click on buttons to be able to upgrade the duration of any power up with their in-game currency (coins). This was outside of any gameplay, as the 'Play' and 'Abilities' options on my Play menu are separate. Therefore, the user is able to upgrade power ups outside the playing environment. Therefore, this success criteria has been fully met.</p>
Ability to pause in-game	42	Partially met	<p>Test No. 42</p> <p>In the final iteration video, the first attempt at pausing the game causes the gameplay itself to stop, but the pause menu user interface does not appear. The game has technically paused, but it has not executed the full subroutine for pausing. After a second click, the pause menu then appears, and in any games after that one, only 1 click is needed to both pause the game and show the pause menu UI at the same time. Due to the fact that the first attempt at pausing does not work, but the rest do, this criteria was partially met.</p>
Maintain current score after coming out of	43	Fully met	<p>Test No. 43</p> <p>As shown in the final iteration</p>

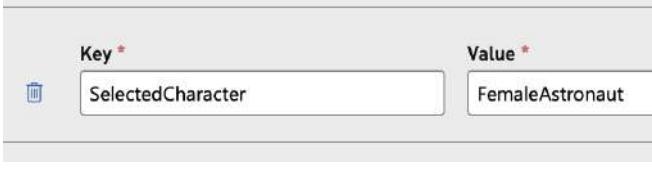
pause menu			video, after pausing and unpausing, the same score is maintained. Towards the end of the final iteration video, the player had collected 70 coins, paused, and when unpause, they still had 70 coins. Therefore, this success criteria was fully met.
Maintains player's state (alive/dead) after pause	44	Fully met	Test No. 44 As shown in the final iteration video, after pausing and unpausing, the player state is maintained. Towards the end of the final iteration video, the player was alive, paused, and when unpause, they were still alive. Therefore, this success criteria was fully met. Therefore, this success criteria was fully met.
Maintains current coins collected after pause	45	Fully met	Test No. 45 As shown in the final iteration video, after pausing and unpausing, the same amount of coins is maintained. In the video, the player had collected 0 coins, paused, and when unpause, they still had 0 coins. Therefore, this success criteria was fully met.
Maintains power up remaining duration after pause	46	Fully met	Test No. 46 As shown in the final iteration video, after pausing and unpausing, the remaining duration of the power up is maintained. In the video, the player had double score active, paused, and when unpause, they still were able to continue using this power up for a long period of time, the rest of its duration. Therefore, this success criteria was fully met.
Player death when player leaves the view	47	Fully met	Test No. 47 In the final iteration video, there are 2 examples of when the player dies when they leave the view. This happens when the player flies out of the top of the screen, and when they do not jump and are pushed out of the left side of the screen.

			These things both killed the player and took them to the game summary menu. The user can only be taken to the game summary menu through the 'PlayerDeath' subroutine, therefore, the player does die when they leave the main camera view. Therefore, this success criteria was fully met.
Player death after collision with obstacle	48	Fully met	<p>Test No. 48</p> <p>In the final iteration video, there is an example of the player dying after colliding with a red triangle, which is the representation of the obstacles in my game. The player collides with the red triangle and is then taken to the game summary menu. The user can only be taken to the game summary menu through the 'PlayerDeath' subroutine, therefore, the player does die when they leave the main camera view. Therefore, this success criteria was fully met.</p>
Random generation of power ups	49	Fully met	<p>Test No. 49</p> <p>In the final iteration video, throughout the gameplay, many power up icons were spawned. There is no clear pattern to the specific power up spawning throughout the gameplay, as there are moments where the same power up will spawn multiple times in a row, or not at all in a single game. This is completely random, and in my code the generation has been designed to pick a random power up from the array of power up names. Therefore, this success criteria was fully met.</p>
After player death, in-game statistics shown in the game summary	50	Fully met	<p>Test No. 50</p> <p>In my final iteration video, after the player dies in my game (e.g. by colliding with an obstacle) the current score and coins collected for that moment were displayed on the screen in big boxes clearly. For example, in one video, the player dies with a score of 1570,</p>

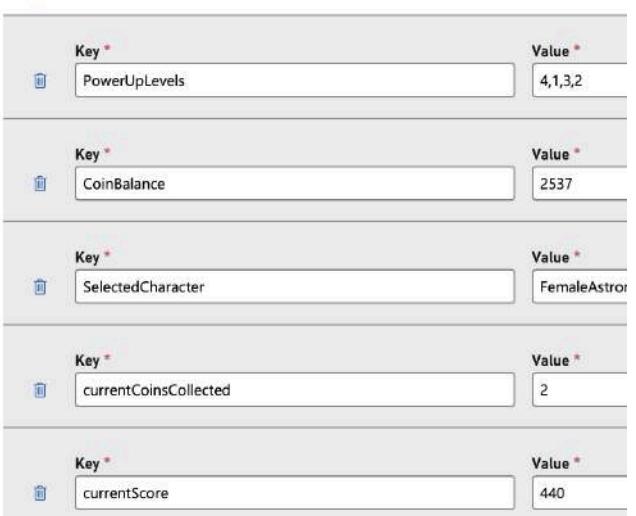
			and 6 coins collected. This information was consistent from the gameplay to the game summary menu. Therefore, this success criteria was met.						
After player death, in-game statistics transferred to external database	51	Fully met	<p>Test No. 51</p> <p>In my final iteration video, after the player dies in my game (e.g. by colliding with an obstacle) the current score and coins collected for that moment were displayed on the screen in big boxes clearly. For example, in one video, the player dies with a score of 1570, and 6 coins collected. This value of 6 was transferred to the external database and added to the current value of coin balance. It is very unlikely to collect a large amount of coins in one game, and so the accumulation of coins (as shown below) is only possible through the playing of multiple games. Therefore, after a player dies, their coins collected must be transferred to an external database. Therefore, this success criteria has been fully met.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: left;">Key *</th> <th style="text-align: right;">Value *</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">CoinBalance</td> <td style="text-align: right; padding: 5px;">429</td> </tr> <tr> <td style="height: 20px;"></td> <td style="height: 20px;"></td> </tr> </tbody> </table>	Key *	Value *	CoinBalance	429		
Key *	Value *								
CoinBalance	429								

7.3.3 - External Database

Success Criteria	Criteria No.	Fully Met, Partially Met or Not Met?	Evidence
Stores highscores	52	Not met	As there is no functionality to do with highscores (was not anything to do with highscores in the video either), this could not be met. There is no mention of highscores within my final iteration, and therefore no existing field in my external database to store them
Stores coin balance	53	Fully met	There is a field within my external database that holds the current

			<p>coin balance of the user. It is also evident that this criteria was met in the final iteration video, as there was a login for a pre-existing user, and when they entered the abilities menu, their current coin balance loaded.</p> <p>The field from PlayFab is shown here:</p> 
Stores selected character	54	Fully met	<p>There is a field within my external database that holds the current selected character for the user. It is also evident that this criteria was met in the final iteration video, as there was a login for a pre-existing user, and when they entered the abilities menu, their previously chosen character loaded.</p> <p>The field from PlayFab is shown here:</p> 
Stores usernames from sign up	55	Fully met	<p>There is a field within my external database that holds the usernames of all users. It is also evident that this criteria was met in the final iteration video, as there was the logging in of a pre-existing user through PlayFab, which would only be possible if usernames were stored from the sign up.</p> <p>The field from PlayFab is shown here:</p> <p>PlayFab username</p> 
Stores passwords from sign up	56	Fully met	<p>There is a field within my external database that holds the passwords</p>

			<p>of all users. It is also evident that this criteria was met in the final iteration video, as there was the logging in of a pre-existing user through PlayFab, which would only be possible if passwords were stored from the sign up.</p> <p>Due to the privacy settings on PlayFab, passwords of users themselves are not visible. However, they are able to change through this function:</p> <p><input type="checkbox"/> Change password</p> <p><input type="checkbox"/> Send password reset email</p>
Stores emails from sign up	57	Fully met	<p>There is a field within my external database that holds the usernames of all users. It is also evident because there is a field in PlayFab to hold the email used to login.</p> <p>The field from PlayFab is shown here:</p> <p>PlayFab login email gmail@gmail.com</p>
Stores data from 1 user all in the same place	58	Fully met	<p>This is evident from a screenshot from a single player's data in PlayFab, where all of their data is stored in the same location:</p>

			<p>Player data</p>  <table border="1"> <thead> <tr> <th>Key *</th> <th>Value *</th> </tr> </thead> <tbody> <tr> <td>PowerUpLevels</td> <td>4,1,3,2</td> </tr> <tr> <td>CoinBalance</td> <td>2537</td> </tr> <tr> <td>SelectedCharacter</td> <td>FemaleAstro</td> </tr> <tr> <td>currentCoinsCollected</td> <td>2</td> </tr> <tr> <td>currentScore</td> <td>440</td> </tr> </tbody> </table>	Key *	Value *	PowerUpLevels	4,1,3,2	CoinBalance	2537	SelectedCharacter	FemaleAstro	currentCoinsCollected	2	currentScore	440
Key *	Value *														
PowerUpLevels	4,1,3,2														
CoinBalance	2537														
SelectedCharacter	FemaleAstro														
currentCoinsCollected	2														
currentScore	440														
Storage of power up levels for each player	59	Fully met	<p>There is a field within my external database that holds the levels of each power up for the user. It is also evident that this criteria was met in the final iteration video, as there was a login for a pre-existing user, and when they entered the abilities menu, their previously upgraded power ups loaded.</p> <p>The field from PlayFab is shown here:</p>  <table border="1"> <thead> <tr> <th>Key *</th> <th>Value *</th> </tr> </thead> <tbody> <tr> <td>PowerUpLevels</td> <td>4,1,3,2</td> </tr> </tbody> </table>	Key *	Value *	PowerUpLevels	4,1,3,2								
Key *	Value *														
PowerUpLevels	4,1,3,2														
Default power up level 1 (Duration 30s)	60	Fully met	<p>There is a field within my external database that holds an array of the levels of each power up for the user, which, when first created, is set to 1,1,1,1. It is also evident that this criteria was met in the final iteration video, as there was a sign up for a new user, and when they entered the abilities menu, their power up levels were set to 1.</p> <p>The field from PlayFab for a new player is shown here:</p>												

			<table border="1"> <thead> <tr> <th>Key *</th><th>Value *</th></tr> </thead> <tbody> <tr> <td>PowerUpLevels</td><td>1,1,1</td></tr> </tbody> </table>	Key *	Value *	PowerUpLevels	1,1,1
Key *	Value *						
PowerUpLevels	1,1,1						
Default coin balance of 0	61	Fully met	<p>There is a field within my external database that holds the coin balance for the user, which when first created, is set to 0. It is also evident that this criteria was met in the final iteration video, as there was a sign up for a new user, and when they entered the abilities menu, their power up levels were set to 1.</p> <p>The field from PlayFab for a new player is shown here:</p> <table border="1"> <thead> <tr> <th>Key *</th><th>Value *</th></tr> </thead> <tbody> <tr> <td>CoinBalance</td><td>0</td></tr> </tbody> </table>	Key *	Value *	CoinBalance	0
Key *	Value *						
CoinBalance	0						
Default highscore of 0	62	Not met	<p>As there is no highscore functionality within my game and there is no reference to it within my code or my external database, consequently, there is no setting of a default highscore to zero, therefore, this criteria was not met.</p>				

7.4 - Opportunities for Further Development

7.4.1 - Game Functionality

This table describes how each partially met and not met criteria can be met in further development in the future, specifically for the functionality of the game.

Success Criteria	Criteria No.	Partially Met or Not Met?	How Can They Be Met?
Clearly labelled menus, with options within them being clear	2	Partially met	By changing the label of the Play Menu to say something that more accurately reflects its contents. This could be a label such as 'Game Hub' or 'Options'

Tutorial Screen UI	7	Partially met	<p>It could be changed to be more interactive, as it is currently just a menu that shows the controls, and isn't really a tutorial, more like instructions.</p>
Back buttons on all menus	12	Partially met	<p>There needs to be an option to go back on the main menu, which allows the user to quit the game.</p> <p>This can be done by inserting a button that says 'Quit' on the main menu in the top left corner, and adding to the main menu manager code some code that makes the button close the application.</p>
Varied environments, with users being able to play in places with different backgrounds	14	Not met	<p>This can be met in 2 ways</p> <ul style="list-style-type: none"> - Either the background changes after a certain amount of seconds in the actual gameplay after a certain amount of seconds to indicate the progression into a harder section of the game - Or, there is another menu added to the Play menu that allows the user to select what background or environment they would like to play in
Implementing challenges to gain in-game currency	27	Not met	<p>This can be added in a separate menu on the Play Menu, in which all challenges are shown. Then in the gameplay, the conditions for these challenges are checked for constantly. Then if a challenge is completed, then the user should be alerted of this in the game summary, have the coins added to their balance, and in the challenges menu, it should be shown that the specific challenge was completed.</p> <p>The challenges should also change after a set period of time, so that the user doesn't run out of challenges to complete.</p>
Indication of power up remaining duration when active	31	Not met	<p>This can be done by implementing a visual representation (e.g. a filled bar that slowly goes down as time passes) that is shown next to the name of the power up, and adding code that makes it work.</p>

Different game modes	35	Not met	This can be met by implementing the hard mode that I had originally planned in my design. This was only not possible because of time restrictions.
Implementation of negative power ups in a harder mode	36	Not met	This can be met by implementing the hard mode that I had originally planned in my design. This was only not possible because of time restrictions.
Implementation of a sharp escalation in difficulty in the harder mode, in comparison to the easy mode	37	Not met	This can be met by implementing the hard mode that I had originally planned in my design, and making the acceleration value higher than that in the Easy Mode. This was only not possible because of time restrictions.
Random generation of in-game currency within the playing environment	39	Partially Met	This can be met by changing the time period between spawns randomly. There could be extra code implemented within the coin spawner script, which picks a random number in a specified range each time a coin is generated, which is the amount of time before the next coin is spawned. This would guarantee the generation of coins is truly random.
Ability to pause in-game	42	Partially met	This can be fully met if the need to couple click on the pause menu was removed. This can be done by rearranging the code so that the pause menu UI is set active before the time scale is changed to be zero.
Stores highscores	52	Not met	This can be met by creating a new key (field) in PlayFab called 'highscore' and adding code that retrieves the current value in that field, compares it to the score in the most recent game, and stores the higher value in the database.
Default highscore of 0	62	Not met	This can be met by creating a new key (field) in PlayFab called 'highscore' and setting it to be zero for a new player when they first sign up (before any games are played)

7.4.2 - Usability Features

Menu/Section	Feature	Partially Met or Not Met	How Can It Be Fully Met?
Main Menu	Simple layout with distinct sections	Partially Met	The buttons 'Login' and 'Sign Up' should be bigger as they are the main focus of the menu. When the buttons are made bigger, the text on them should also be made bigger, so they stand out.
	Title of Menu Clearly Labelled	Partially Met	The title should be changed to include 'Main Menu' in some way. Such as being called 'AstroSprint: Main Menu', so that it is extremely clear to the user that it is the main menu.
Login Menu	Buttons clearly labelled	Not Met	Make the login button and its text bigger so it is easier to read, and make it slightly further from the input fields, so there is a no accidental clicking of the login button
	Consistent, readable fonts	Partially Met	The font on the actual 'log in' button is not very readable, so it should be changed to be larger so it is easier to read.
	Visible feedback when hovering over buttons	Not Met	Add a feature where the login button becomes visibly darker, with a noticeable change, when the mouse is hovering over it
Sign Up Menu	Clear error messages on inputs entered	Partially Met	Change the font size of the messages to be slightly smaller so that the message that talks about the email format can have all of its words be on

			continuous lines, without them being broken up like they currently are.
	Buttons clearly labelled	Partially Met	Make the actual sign up button bigger so it's easier to read and therefore clearer (which by extension means the actual sign up button should be bigger.)
	Visible feedback when hovering over buttons	Partially Met	Change the visible feedback so it is darker than it currently is, so the change is noticeable.
	Consistent, readable fonts	Partially Met	The text on the sign up button is hard to read, so it should be made bigger so that it is readable.
Play Menu	Title of menu at top of screen	Not Met	The name of the Play Menu should be changed so that it says something more clearly related to the function of the menu, such as 'Options' or 'Game Hub'
	Visible feedback when hovering over buttons	Not Met	Add a function to the button so that the buttons on the play menu become darker when the mouse is hovered over them, and make it dark enough so it is clear
Controls Screen [Previously Tutorial Screen]	Visible feedback when hovering over buttons	Partially Met	There should be a function added to the back button so that it becomes visibly darker when the mouse is hovered over it.
Character Menu	Consistent, readable fonts	Partially Met	The fonts on the select/selected buttons should be changed to be in the 'Courier New' font, so they are consistent with the

			whole menu.
	Visible feedback when hovering over buttons	Not Met	There should be a function added to all of the character selection buttons where they get slightly darker when the mouse hovers over them.
Abilities Menu	Buttons disabled when maximum level of upgrade has been reached	Partially Met	The pause menu subroutine code should be moved around so that the code attached to the pause button that activates the pause menu UI is activated before the time scale is changed, so that it doesn't require 2 clicks to active the UI
	Visible feedback when hovering over buttons	Not Met	There should be a function added to all of the price upgrade buttons where they get slightly darker when the mouse hovers over them.
Gameplay	Name of current power up labelled	Partially Met	Change the code so that the collecting of a power up should instantiate its own power up text, in which the UI is checked for the next free space. This means that the names of the power ups will never overlap, while still showing the names of all the currently active power ups
	Power up icons look like power ups, and icons clearly reflect what they are	Partially Met	Change the flying icon to be a different colour and make the word 'Fly' bigger. Change the double jump icon to add a '2x' or 'Double' to indicate it being a power up where you can jump twice.

Pause Menu	Visible feedback when hovering over buttons	Not Met	There should be a function added to all of the buttons on-screen, where they get slightly darker when the mouse hovers over them.
Game Summary Menu	Visible feedback when hovering over buttons	Not Met	There should be a function added to all of the buttons where they get slightly darker when the mouse hovers over them.

7.5 – Maintenance & Limitations

7.5.1 – Maintenance Issues

Due to the fact that my code is split into different scripts attached to different game objects, it is very modular. This modularity means that individual scripts can be worked on without affecting the functionality of other scripts. However, there may be potential issues for the longer scripts that contain multiple different functions for the game, for example, the player script is extremely long and contains the functionality of sending data to an external database, which could easily be done in a separate script.

The long length of this script means that it may be difficult to add to in the future, so this may become an issue, as too many functions are put into one script.

Another issue with this game is the naming of the files, as some of the files in Unity, specifically image files, are not named as clearly as they can be. Some of them are named things such as 'foreground1' to represent foreground, but this has no indication of what this is the foreground of. The lack of good naming conventions in some of these files may cause issues when trying to develop the game further.

Currently, the game lacks the ability to be able to report an issue with the game back to the developers, or to be able to reset passwords to user accounts. This would need to be addressed if this game were to be commercially distributed.

7.5.2 – Limitations of Solution

Listed below is a table of all of the limitations of my current solution, and how they would be beneficial to the development of AstroSprint.

Requirement	Justification
-------------	---------------

Multiple game modes with different controls	The current version of AstroSprint only has 1 game mode, with vertical motion controlled by the player. The implementation of different game modes with different controls would make the game more engaging by giving the user more options to choose from, so they don't get bored by playing the game gameplay repeatedly.
Expansion of character catalogue	The current catalogue of characters only consists of 4 characters. Adding more characters would mean that the user has more options to choose from.
Horizontal movement controlled by player	The players currently can only control the vertical movement of their character. Adding this would make the game more interesting, and possibly easier, for players, as they would have more control over their character and what they do.
Highscore leaderboards	Users can currently only see their score after a single game. Adding this would add a strong sense of competition if players were able to see how they rank compared to others who play the game. Would encourage players to play more and become better at the game in order to top the charts.
More types of obstacles	The implementation of different types of obstacles (e.g. obstacles that follow the player) would make the game more challenging and therefore more engaging.
Colourblind filters	The game currently does not have obvious accommodation for those with colour blindness, apart from using contrasting colours. The implementation of filters available for users to turn on to filter the colours to make them easier to see. This would also make the game more accessible.
Music and audio feedback	The game is currently silent. The implementation of music and audio feedback would make the game more engaging for the user, and audio feedback specifically would make the game more accessible to those with vision impairments.
Interactive gameplay tutorial	The game currently only has a controls menu. The inclusion of an interactive gameplay tutorial would make the game easier to understand for first-time players, and could give them key information about obstacles, coins and power ups. This would also make the game more usable.

Implementation of highscores and storing of them	The game currently does not have any functionality to deal with the highscores, or storing them. I planned to implement this, but was not able to due to lack of time. Storing highscores could mean that the player has an opportunity to try and beat themselves at the game and will make it more engaging.
Implementation of challenges to gain in-game currency	This was something I wanted to add in my success criteria, but was not able to due to the lack of time. This, however, would make the game more engaging, as it would give the players something to work towards and make the game more interesting and engaging.

7.6 – Opportunities to Minimise Limitations

7.6.1 – Potential Development to Deal with Limitations

Requirement	Opportunities to Implement
Multiple game modes with different controls	This could be done by creating a new button on the Play Menu to take the user to a different game mode entirely. In these new game modes, new background and foreground designs can be made, and the keybinds within each game mode can be made unique. There could be a game mode in which the foreground is upside down, and the player jumps down, away from the foreground (which is now on the ceiling), which would have the same keybinds as the game does currently, except it does a Vector3.down translation instead of a Vector3.up translation. This would be simple to implement as the functionality for the most part would be the same, just in the opposite direction.
Expansion of character catalogue	This could be done by redesigning the menu to incorporate 6 characters on one screen, and have multiple scrollable sections that the user can choose from. This would mean also finding new character designs, which can be done by using other space emojis, and possibly designing new characters online.
Horizontal movement controlled by player	This can be done by altering the 'Update' function to include the detection of the left and right arrow key codes. The direction of the arrow should correspond to a Vector3 translation in the x-axis. These changes would allow the user to move horizontally.

Highscore leaderboards	If highscores were stored, it would be possible to retrieve all of the highscores of the users, and order them based on size. The top 10 highest scores, along with the usernames of the users who achieved them, would be put onto a new menu in the Play Menu called 'Leaderboard', to show the best people at the game.
More types of obstacles	This could be implemented by making 2 more obstacle prefabs, each with a different colour, and put them (and the original obstacle) in an array. A random number function could be used to generate an index of which obstacle to spawn each timer interval. These new power ups could each have their own scripts, and do things such as follow the player for a set period of time, or come up from the bottom of the screen, rather than falling from the top.
Colourblind filters	This could be done by implementing a settings menu (available from the Play Menu or Main Menu) which would have a setting where you can turn on/off colour blind filters in-game. This will put an overlay over the game which will make it easier and more accessible for colour blind users to play.
Music and audio feedback	Music can be added to the gameplay by adding a new empty game object into the gameplay scene, uploading an audio into the assets folder, and assigning this to the empty game object, setting the audio to loop. Audio feedback can be added to buttons by importing the audio file, creating a function that plays the audio, and adding this function to the 'On Click' of the button in the Unity Inspector.
Interactive gameplay tutorial	Implement a separate menu, accessible by the Play Menu, which is a slower, more controlled version of the gameplay, with all of the random aspects removed. This would mean that each object in the actual gameplay (coins, obstacles & power ups) will all be spawned in turn, and have text pop up on-screen to explain to the user what each of them are, and what happens after colliding with them.
Implementation of highscores and storing of then	This was something I had planned to do, but did not have time. There would be a field in PlayFab that is for holding the highscores. For all new players, this would be defaulted to be zero. After every single game is played, the highscore is retrieved from the external database, and compared to the final score of the current game. If the current score is greater than the highscore, the current score

	is set as the new highscore, and the player is alerted that they have set a new highscore.
Implementation of challenges to gain in-game currency	<p>This could be done by the creation of a new option in the Play Menu, which would say 'Challenges'. After clicking on this button, the user would be taken to the challenges menu, which would display the following things:</p> <ul style="list-style-type: none"> - Name of challenge - Progress made towards challenge - Coins that will be gained on completion <p>During gameplay, it will be inserted in the 'Update' function, a subroutine to check if the challenge has been completed.</p>

7.6.2 - Potential Development to Deal with Potential Changes

To deal with potential changes, if the game is commercially distributed, there should be automatic updates for all of the distributed copies. When more features are added to the game, they should be thoroughly tested by the developer and the primary user, prior to slowly being released to larger audiences before the update goes out to every user that has a copy of the game.

There should also be an option for users to be able to report a problem with the game, whether that is inside of the game through a feedback system, or externally through a contact email or website.

After any features have been implemented, as well as testing the new features itself, there should be thorough regression testing to ensure that the old features of this game are not affected by the new changes.

8 - Bibliography

Microsoft PlayFab Learning Platform: Quickstart

- <https://learn.microsoft.com/en-us/gaming/playfab/features/playerdata/quickstart>
- <https://learn.microsoft.com/en-us/gaming/playfab/sdk/c-sharp/quickstart>
- Used this to see the different PlayFab functions and see examples on how to use them, not code directly taken.

Game Backend

- <https://www.gamebackend.dev/getting-started-with-playfab-in-unity-c86c3a0381f7>
- Used this to learn how to set up PlayFab in Unity

Unity Documentation

- <https://docs.unity3d.com/Manual/index.html>
- Used this to see the different functions in Unity and how to use them

Youtube Tutorial: Main Menu

- <https://www.youtube.com/watch?v=-GWjA6dixV4>
- Used this to get an idea of how to make a main menu, took ideas from this video, but did not have the same options on my menu as shown in the video, so this was only used as inspiration.

Youtube Tutorial: Setting up Unity with PlayFab

- <https://www.youtube.com/watch?v=Zuh0ZjfUbvA>
- Used this to help me to fully understand the process of linking PlayFab to Unity, as it is a narrated guide

Google Forms

- <https://docs.google.com/forms>
- Used to create surveys to get stakeholder feedback in the analysis, and get responses from the primary user on the usability features

Google Slides

- <https://docs.google.com/presentation>
- Used to design all of the menus, the foreground and all the buttons

Coolors.co

- <https://coolors.co/>
- Used to create the colour palettes shown in my Design phase.

Oxford Languages (Dictionary)

- <https://languages.oup.com>
- Used to define the words 'Tutorial' and 'Controls'

Game Rant

- <https://gamerant.com/games-released-steam-2023-record-number/>

Software Development Methodology Image (Introduction)

- <https://pmt.physicsandmathstutor.com/download/Computer-Science/A-level/Notes/OCR/1.2-Software-and-Software-Development/Advanced/1.2.3.%20Software%20Development.pdf>

Draw.io

- <https://draw.io>
- For the creation of flowcharts

Icon Images:

Icon	Name	Use	Image Source
------	------	-----	--------------

	Double Jump	A representative of the double jump power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.rolimons.com/gamepass/9230676
	Double Points	A representative of the double points power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.roblox.com/game-pass/11393260/PRE-RELEASE-Double-Points
	Fly	A representative of the flying power up, to be used in the Easy Mode & Hard Mode gameplay, as well as the abilities menu	https://www.roblox.com/game-pass/2596638/Fly
	Double Coins	A representative of the double coins power up, to be used in the Easy & Hard Mode gameplay and the abilities menu	https://www.roblox.com/ko/game-pass/103796180/Gamepass-Icon
	Male Astronaut Character	A representative of the male astronaut character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/2010/smileys-people/man-astronaut
	Female Astronaut	A representative of the female astronaut character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/2106/smileys-people/woman-astronaut
	Star Character	A representative of the star character, to be used in the Easy & Hard Mode gameplay and the	https://www.emoji.com/view/emoji/586/animals-nature/white-medium-star

		character menu	
	Alien Character	A representative of the alien character, to be used in the Easy & Hard Mode gameplay and the character menu	https://www.emoji.com/view/emoji/34/smiley-people/alien
	Darkness	A representative of the darkness power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/close-eyes_4264667
	Blurry Screen	A representative of the blurry screen power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/dizzy_2302669
	Half Score	A representative of the half score power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/fraction_9845747
	Inverted Colours	A representative of the inverted colours power down, to be used in the Hard Mode gameplay	https://www.flaticon.com/free-icon/invert_7746578
	New Highscore Icon	Appears on the game summary screen when a new high score is achieved	https://www.flaticon.com/free-icon/high-score_13446444
	Back Button	The button that allows players to go backwards to a previous menu/screen in-game	https://www.flaticon.com/free-icon/back_318292