Cartoon character of a small alien

Description automatically generated with medium confidence

**Video:**

<https://youtu.be/OQx_68OmVFY>

3D Platformer Game

By Johan Jobi

**CENTRE NUMBER: 57117**

**CANDIDATE NUMBER: 8149**

Table of Contents

[Analysis 4](#_Toc163824134)

[Problem 4](#_Toc163824135)

[Development Platform 4](#_Toc163824136)

[Unreal Engine 4](#_Toc163824137)

[Unity 4](#_Toc163824138)

[Solution 4](#_Toc163824139)

[Research 5](#_Toc163824140)

[Portal / Portal 2 5](#_Toc163824141)

[Super Mario Odyssey 6](#_Toc163824142)

[It takes two. 6](#_Toc163824143)

[Human: fall flat 7](#_Toc163824144)

[Gary’s Mod 8](#_Toc163824145)

[Interview Questions 8](#_Toc163824146)

[Questionnaire Results 9](#_Toc163824147)

[Level Design 15](#_Toc163824148)

[Database Design 15](#_Toc163824149)

[SuccessCriteria 16](#_Toc163824150)

[Documented Design 17](#_Toc163824151)

[Procedural Generation 17](#_Toc163824152)

[Binary Space Partitioning 17](#_Toc163824153)

[Wave Collapse Function 17](#_Toc163824154)

[Grappling Ability 18](#_Toc163824155)

[Hierarchy Diagrams 22](#_Toc163824156)

[Options Menu 22](#_Toc163824157)

[Main Menu 22](#_Toc163824158)

[Movement Script 23](#_Toc163824159)

[Enemies 26](#_Toc163824160)

[Nav Mesh Agent 28](#_Toc163824161)

[Maze Generation 31](#_Toc163824162)

[Inventory 34](#_Toc163824163)

[Items 34](#_Toc163824164)

[Database 35](#_Toc163824165)

[User Interface 36](#_Toc163824166)

[Inventory Object 36](#_Toc163824167)

[Billboard 37](#_Toc163824168)

[Saving and Loading files 37](#_Toc163824169)

[Login System 39](#_Toc163824170)

[PlayFab API 39](#_Toc163824171)

[PlayFab 40](#_Toc163824172)

[Hashing 42](#_Toc163824173)

[Testing and Debugging 45](#_Toc163824174)

[Movement Script 45](#_Toc163824175)

[Throwing Behaviour 46](#_Toc163824176)

[Grappling 46](#_Toc163824177)

[Maze Generation 47](#_Toc163824178)

[Enemy 48](#_Toc163824179)

[Menu 48](#_Toc163824180)

[Inventory 49](#_Toc163824181)

[Login System 51](#_Toc163824182)

[HashMap 52](#_Toc163824183)

[Hashing Algorithm 52](#_Toc163824184)

[Technical Solution 53](#_Toc163824185)

[Movement Script 53](#_Toc163824186)

[Background Music 58](#_Toc163824187)

[Billboard 59](#_Toc163824188)

[Button 59](#_Toc163824189)

[Combat Camera 60](#_Toc163824190)

[Combat Cam Toggle Button 61](#_Toc163824191)

[Default Object 62](#_Toc163824192)

[Display Score 63](#_Toc163824193)

[Dynamic Interface 64](#_Toc163824194)

[Enemy 1 66](#_Toc163824195)

[Equipment object 69](#_Toc163824196)

[Food Object 69](#_Toc163824197)

[Grappling 71](#_Toc163824198)

[Ground Item 73](#_Toc163824199)

[Hashmap 74](#_Toc163824200)

[Health Bar 77](#_Toc163824201)

[Item Database Object 77](#_Toc163824202)

[Item Object 78](#_Toc163824203)

[Item Spawn Button 79](#_Toc163824204)

[Inventory Object 79](#_Toc163824205)

[Main Menu 84](#_Toc163824206)

[Maze Cell 85](#_Toc163824207)

[Maze Generation 86](#_Toc163824208)

[Next Level Button 89](#_Toc163824209)

[Number Game Controller 90](#_Toc163824210)

[Number On Button 92](#_Toc163824211)

[Options Menu 93](#_Toc163824212)

[Pause Menu 94](#_Toc163824213)

[Player HP 96](#_Toc163824214)

[Player Inventory 98](#_Toc163824215)

[PlayFab Manager 102](#_Toc163824216)

[Projectile Addon 105](#_Toc163824217)

[Projectile Addon E1 106](#_Toc163824218)

[Spawn Items 107](#_Toc163824219)

[Static Interface 108](#_Toc163824220)

[Switching 110](#_Toc163824221)

[Throwing Behaviour 112](#_Toc163824222)

[User Interface 114](#_Toc163824223)

[Evaluation 119](#_Toc163824224)

[Final Interview 119](#_Toc163824225)

# Analysis

Type: Co-op 3D Platformer Game

## Problem

There are not many platformer games which are strategy based as well as having a shooter aspect. Most games like this have been discontinued for example Jack and Dexter (the latest version of this game does not have the best graphics). Also, games like this have an overwhelming number of features which can be difficult to understand for a player who just started gaming. As well as this, projects like this have a team of developers who hand craft levels which takes time and money.

## Development Platform

### Unreal Engine

Unreal Engine has high quality graphics which utilise real-time ray tracing which is ideal for a game like mine as it is a competitor for games like Jack and Dexter. As well as this, it is known to be used in the film and TV industry. However, because of the high graphics quality it is more resource intensive and may not be suitable to be played on a low-end computer which, my client (a person who has little to no experience with games), is most likely to have.

### Unity

Unity has lower end graphics which can be better as a new gamer may have low performance devices. Also, it has a big asset store so I don’t have to make everything from scratch meaning the client can get their game quicker.

## Solution

Therefore, my client (Alan Thomas) wants such a game with good game mechanics and vector graphics as well as being able to play with his friend in the same network. I researched that using Unity would be optimal for making 3D vector graphics as well as supporting gameplay over LAN. My aim is to incorporate a few key features which would make the game simple but still be entertaining which is appealing for a person who is a beginner to gaming such as my client. I also want to create an automated level generator; this will reduce the time to make levels.

## 

## Research

### Portal / Portal 2

A picture containing screenshot, line, design, art

Description automatically generatedA white rectangular object with a black background

Description automatically generatedPortal is a 2007 puzzle-platform game developed and published by Valve. Portal consists primarily of a series of puzzles that must be solved by teleporting the player's character and simple objects the portal gun, a device that can create portals between two flat planes.

The first picture shows how the magnitude of linear momentum is conserved through the portals. By jumping in the blue portal, the character is launched out of the orange portal on to the platform on the right.

The second picture shows a more advanced portal technique where the player builds up speed using two blue portals two reach the platform on the right. The player jumps through the first blue portal and ends up out the orange portal. As the player comes out of the orange portal a blue portal is created as the player is in the air to build up speed and get to the platform on the right.

Positives

* The graphics and game mechanics are simple and are easy for new gamers to understand.
* It has a good story line and has an original idea.
* The game is a co-op and can be played over the internet.
* All levels are hand crafted, so it makes sense in the story line.
* The game uses accurate physics.

Negatives

* The game has no automatically generated levels which means it takes a long time to make new levels.
* Same game mechanics are repeated making it boring in the long term.
* Took a lot of money to develop.
* No competitive play
* Outdated graphics
* Game is quite short.

### Super Mario Odyssey

Super Mario Odyssey is a platform game developed and published by Nintendo for the Nintendo Switch. An entry in the Super Mario series, it follows Mario and his new ally Cappy—a sentient hat that allows Mario to control other characters and objects—as they journey across various kingdoms to save Princess Peach from his nemesis Bowser's plans of forced marriage. In contrast to the linear gameplay of prior entries, the game returns to the primarily open-ended, 3D platform gameplay featured in Super Mario 64 and Super Mario Sunshine

Positives

* Well-designed 3D vector graphics
* Original game mechanics and idea’s
* The game can be played with two players.
* In depth story line

Negatives

* The game has no automatically generated levels
* The game is not multiplayer over LAN.
* The game took a lot of time and money to develop.
* Camera and movement are not precise.
* A new gamer may not understand the concepts of Mario.
* A new gamer may be overwhelmed with the details of the story line and

### It takes two.

It Takes Two is an action-adventure video game with elements from platform games. It is specifically designed for split-screen cooperative multiplayer, which means that it must be played with another player through either local or online play. The game features many game mechanics from various video game genres. These gameplay mechanics are connected to the story and the theme of the level. For instance, in one level, Cody gains the ability to rewind time, whereas May can replicate herself. Players must cooperate with each other and utilize these abilities in order to progress. The game also features many minigames.

Positives

* Easy for new gamers to understand the concept of the game.
* Although game mechanics change per level, they are easy to understand so a new gamer will not get bored or too overwhelmed by the features.
* You can play with friends over LAN or Split screen.
* Smooth gameplay and accurate physics
* The gameplay affects the story.

Negatives

* No automated levels

### Human: fall flat

Human: Fall Flat is a physics puzzle game where players play a customisable human, referred to in-game as Bob. Bob is stated to have no superhuman abilities; he is purely human. Players can make him grab objects and climb up ledges using both his arms and looking with his head.

Although Bob's standard appearance is a featureless, minimalist all-white human with a baseball cap, players can customise him to their liking, painting his body in a different array of colours and dressing him in a variety of costumes.

The game is open-ended. Each level is themed differently, each containing multiple solutions to their unique puzzles. Various remotes hidden in the game give players clues to learn the gameplay and ultimately solve the puzzles.

Positives

* Good rag doll physics
* You can play with people over the internet.
* Level based.
* Puzzle game.

Negatives

* No automated level generation
* Controls can be frustrating.

### Gary’s Mod

Gary’s Mod is a physics-based sandbox game that has no set objectives so the players can do whatever they want. There is a physics gun in the game which can manipulate how objects can be moved, picked up, rotated and other properties.

Positives

* Uses ragdoll physics and is very realistic.
* Multiplayer

Negatives

* As there are no set objectives a new gamer may be confused on what to do and may get bored easily
* No story line so the game may not make much sense.

### Interview Questions

1. **Have you played many 3D platformer games? What do you like/ dislike about them?**

I like precise and responsive controls that allow for smooth and enjoyable gameplay. I also like the fun and unique abilities to make the game more engaging. However, sometimes the game can be confusing to me if there are too many options or controls. Also, I disliked the unfamiliar game mechanics for example in Super Mario Odyssey as I have never played a Mario game before.

1. **Would you prefer better graphics or better storyline in a 3D platformer game?**

I like both however I prefer having better graphics as a storyline can only be done once but graphics can constantly be changed by the community. Overall, a combination of both gives.

1. **Would you prefer the game to have a linear progression, or do you prefer a more open-world structure?**

I prefer a more linear progression as otherwise there are too many options, some levels however could have an open-world aspect for example more than one way to finish the level so that there is more freedom.

1. **As this project has time restraints, are there any limitations that can be considered?**

The storyline and/or backstories doesn’t have to be very detailed in my opinion, but the game mechanics and abilities of players should be creative and unique. There should only be a few quality levels than many mediocre ones.

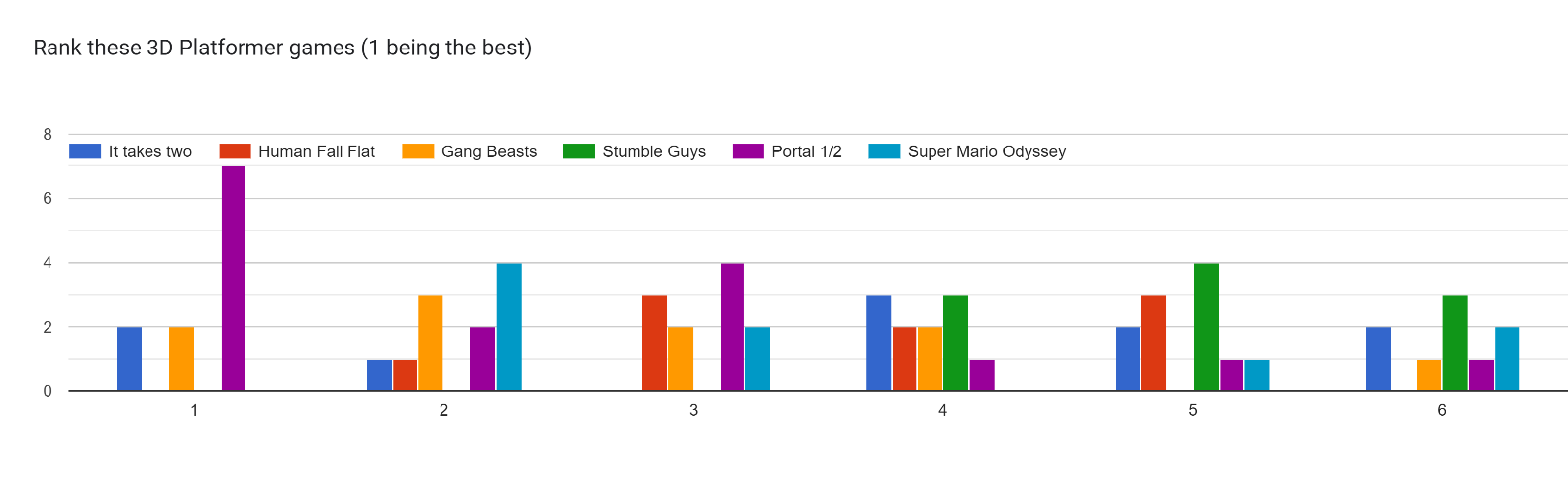
1. **What features would you like to see in the game?**

I would like to see fun game mechanics and abilities/ guns that have special abilities when used together, I think this will make the game more enjoyable.

1. **What parts of the program should be automated?**

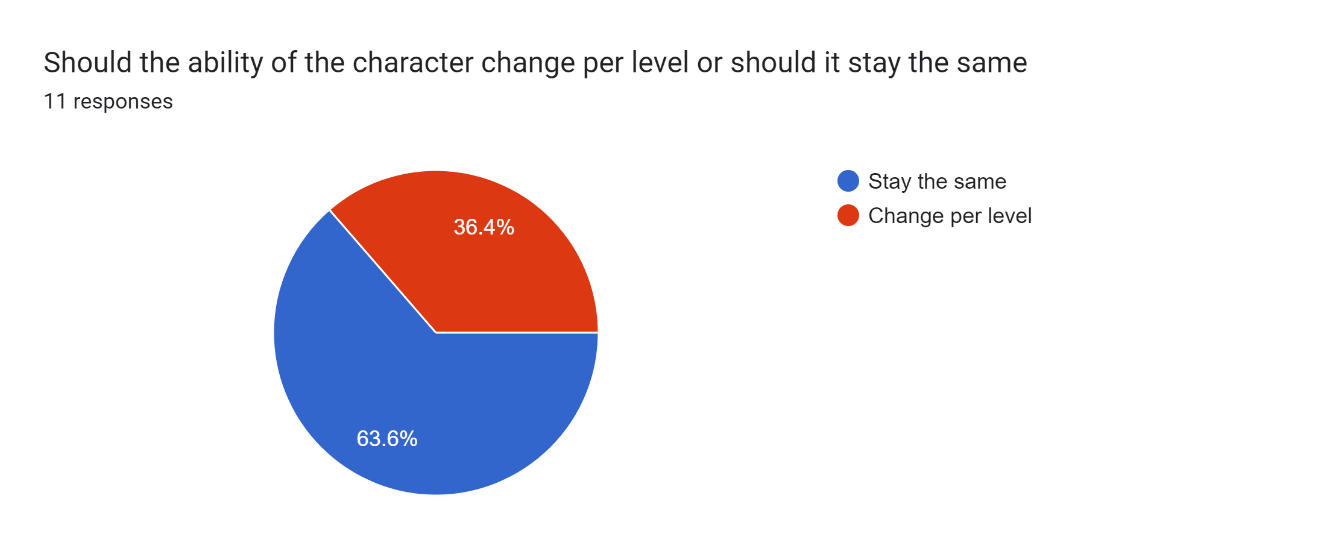
I think enemies could be automated so that it makes the game more fun, this makes it so not every level is just a puzzle solving level.

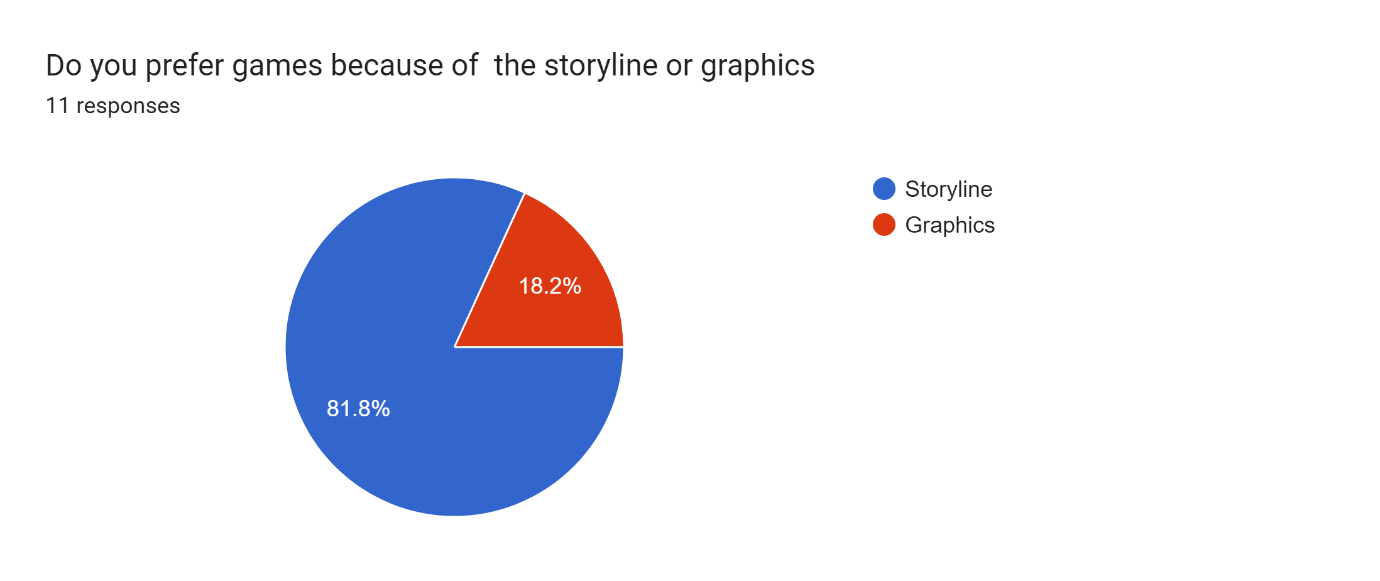
### Questionnaire Results



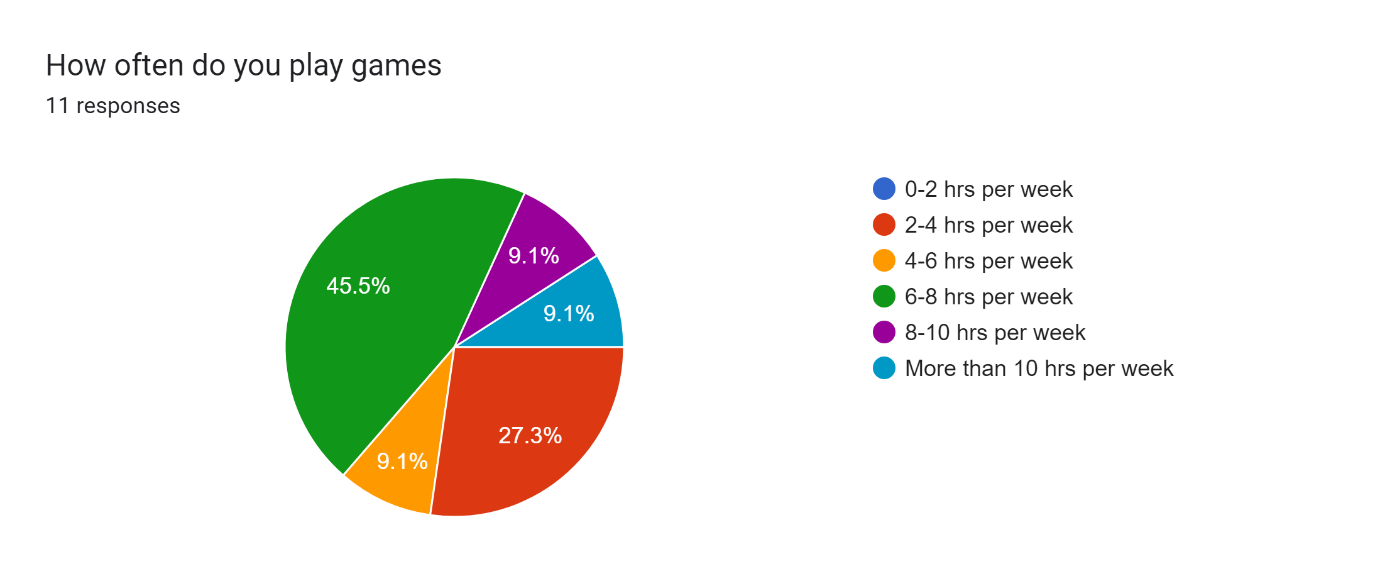
From my questionnaire, most people ranked Portal as the best 3D platformer and Stumble Guys as the worst out of the six games. This suggest users want a 3D platformer with a shooting aspect that can be used to solve puzzles and complete levels with a very basic storyline.

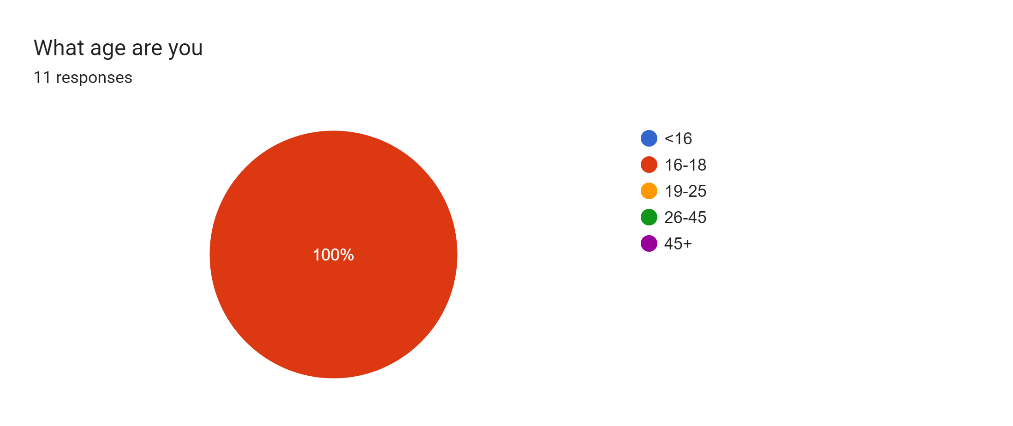
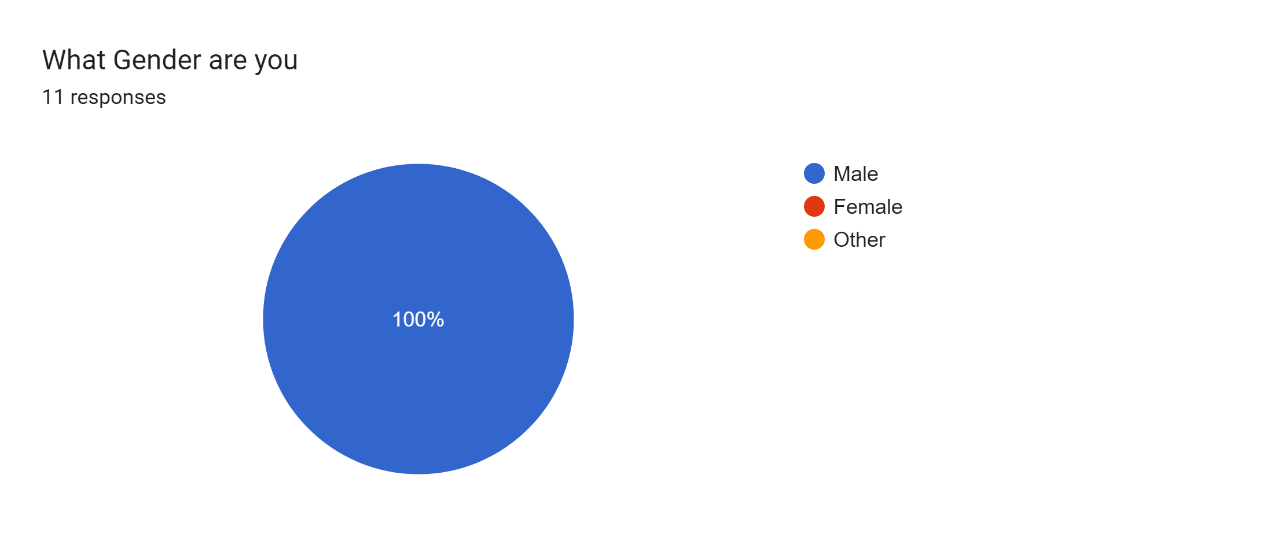
Many people (including my client Alan) wanted the ability of the character to stay the same to per level rather than changing, this means one unique and fun game mechanic is preferred over many mediocre ones.





Although many people said that portal was their favourite game, more people wanted a preferred to have a better storyline than have better graphics. My client disagrees with this and suggests better graphics. However, some people may like a storyline because of the plot-twists in the game which is much less time consuming than making a complex story.



Most people play games 6-8 hours a week which is not my desired audience so most of the results from the survey may not be the best for my client.

A screenshot of a computer

Description automatically generated with low confidence

All the people who took my questionnaire are 16 to18-year-old males, so there isn’t much variety which could take all aspects into account however my client is a 16-year-old male so they potentially may have similar opinions.

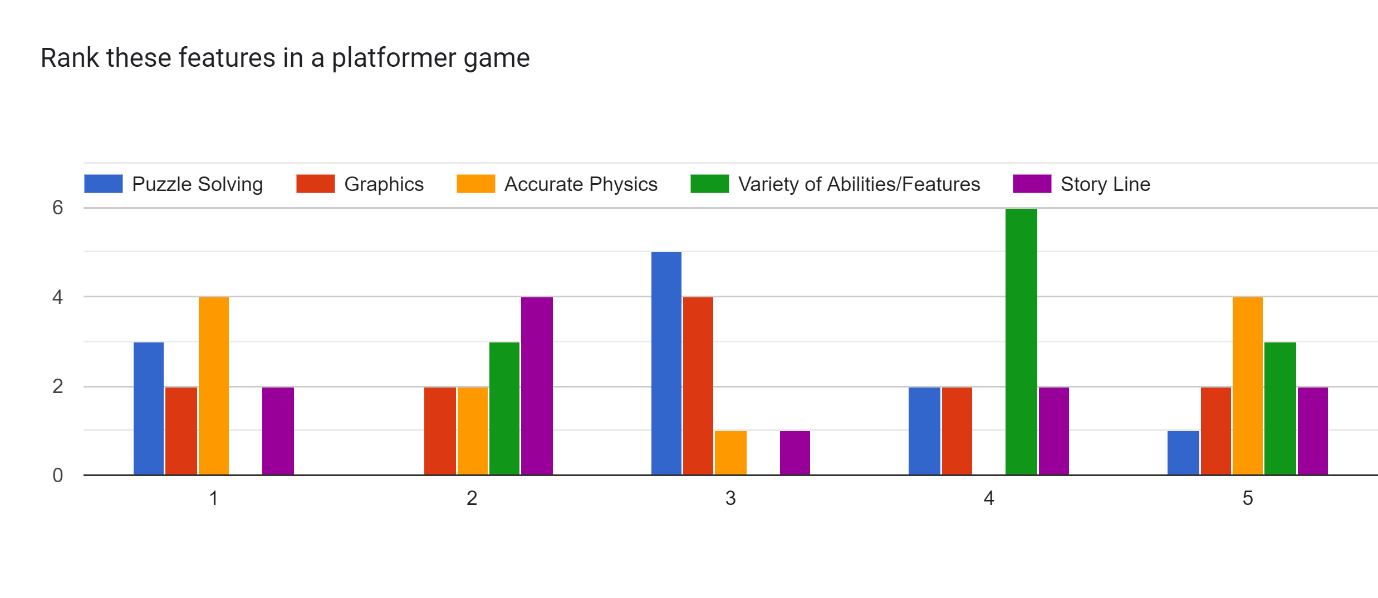
There were many good points for potential limitations, one person said shops and ability to customize characters.

A screenshot of a computer game

Description automatically generated with low confidence

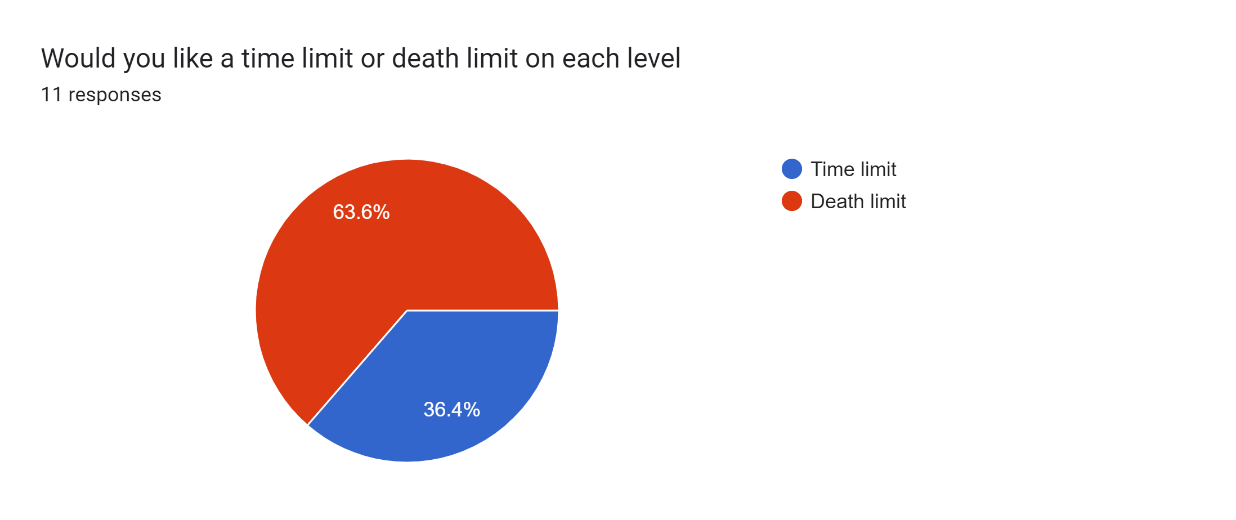
A picture containing text, font, screenshot

Description automatically generatedPeople wanted good physics like in portal gun as well as a variety of well-crafted and unique abilities. Some value graphics, most 3d platformer games do not have realistic graphics so well-crafted cartoony graphics would be more suitable.

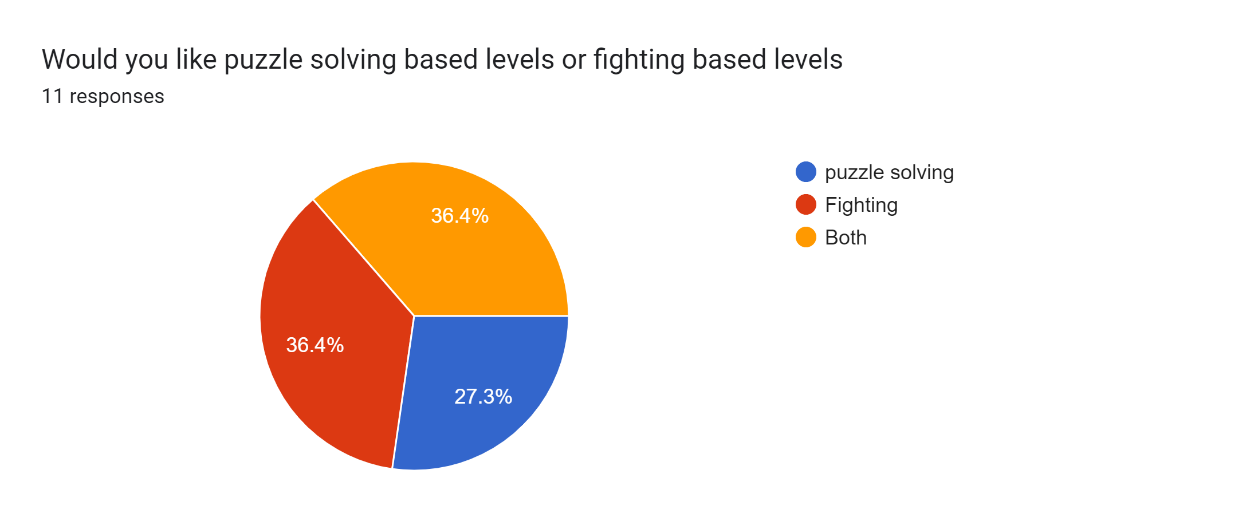
A screenshot of a computer

Description automatically generated with medium confidenceAccurate Physics took the number one spot with a storyline at second, a variety of abilities to a lower spot than I expected as a lot of people thought it was their favourite part, but it took a strong fourth place. Fifth place is not very clear so not many comments can be made.

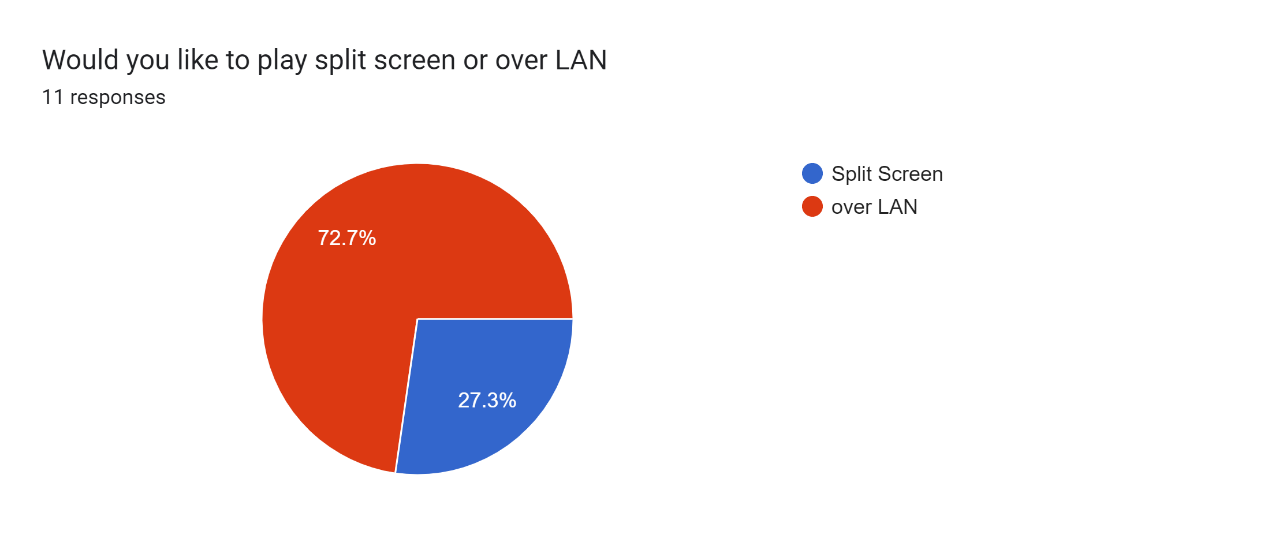
Some good points to take out are abilities such as slowing down time, dash, double jump and telekinesis I think these features are fun to use and easy to understand which is perfect for my client.



Most people want a death limit which means players can take time for completing their levels as long as they are careful which is better for a beginner gamer as they need to be more strategic than be fast with controls. By learning from mistakes every time, they die will create a sense of progression as they are improving from last time.



“Both” and “Fighting” had the same number of votes so I will include both but favour fighting more.



Multiplayer over LAN is more traditional than split screen so LAN will be the way forward.

## Level Design

From this my plan is to have two types of levels. The first type will have two players in different rooms or with different perspectives trying to solve puzzles together. The second type will have a shooting aspect to it where the two players must work together to defeat the enemies.

For the first level, there will be two players who spawn in different rooms and have to work together to select two numbers which add up to get the number that is displayed on the screen. I will also include some parkour here to make this level a bit longer.

For the second level there will be a maze. One player will spawn on top of the maze whilst the other is inside of the maze. One player will have to navigate the other through the maze to get to the end.

The third and final level will consist of a combat game where players must defeat AI enemies and find keys for them to win. The number of enemies they kill will correspond to the final score they achieve.

## Database Design

A diagram of a computer

Description automatically generated

If I choose to use a database to a design may game this is a very simple version/plan of it. Some entities can be vague like controls in settings can have another table linked with all the possible controls and the key that they player wants to map it on to. If I decide to use procedural generation techniques, I can make a base generated Level which may contain many rooms which contain many hand-crafted levels. This is the most suitable solution as my client doesn’t want too much exploration aspects in the game so this should be the perfect balance. Following from procedural generation, the Enemy Attacking function can use A\* pathfinding (or equivalent) to track the player if the level is generated, for example, as a maze.

## SuccessCriteria

1. The user must be greeted with a main menu.
   1. The main menu must contain information on how each level of the game works (in case the player gets confused)
   2. The main menu must contain information about the controls of the game.
   3. The main menu must contain options to change the resolution of the game. These options should contain valid resolutions native to the monitor.
   4. The main menu must contain options to change the graphics quality of the game.
   5. The menu should contain options to change sound.
   6. The menu must have a login system with registering and password resetting.
2. The game should be two player and be playable over LAN.
3. Levels in the game should not be too difficult to understand for someone who is new into gaming.
4. The character should be animated with smooth transitions between each animation state.

**4.1** when pressing any of the movement keys the character must play the walking animation.

**4.2** when pressing the shift key whilst moving, the running animation must play, and the character must move faster.

**4.3** when falling off a platform or jumping down the falling animation must play.

**4.4** when jumping up the jumping animation must play.

**4.5** when the character is still the idle animation must play.

1. The character must be easy to control.

**5.1** The character should be positioned well in the camera.

**5.2** There should be camera modes for different views of the character.

**5.3** There must be crosshair making it easier to aim for shooting levels.

**5.4** The player should move using WASD keys and press and hold shift to run.

**5.5** The character should rotate in the direction the camera is facing when pressing W key to move forward. Otherwise, if the camera is rotating around the character, the character should not move.

1. The first level must be interactive.

**6.1** The first level must contain some aspect of working together.

**6.2** The first level must be a problem-solving level.

**6.3** The first level must have some sort of movement-based challenges e.g. parkour.

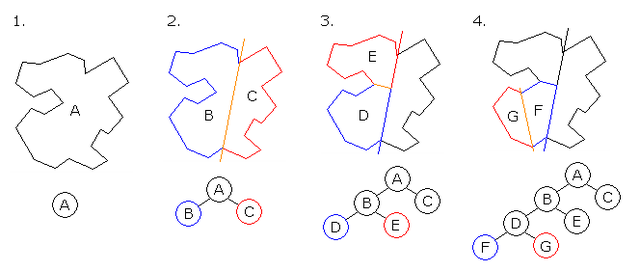
1. The second level must be a randomly generated maze.
2. The third level must consist of a shooting aspect which uses realistic physics.
   1. This level must consist of automated NPC enemies.
   2. This level must give you a score depending on the total amount of enemies killed.
   3. The score must be displayed and saved to a server with a leaderboard.
3. The game must have an interactable inventory system.
   1. It must contain an equipment inventory which contain items that is being used by the character e.g. armour, grapples, guns, food etc.
   2. It must contain a normal inventory where you keep items you are not using.
   3. Each item must be defined by a key in a database or dictionary.
   4. Only specific items can stack in the inventory. Player must have the option to move or remove items in their inventory or between inventories.
4. The game should not be too resource heavy.
5. The game should have fun abilities with accurate physics using SUVAT calculations.
6. All aspects of the game must be simple to use complete.

# Documented Design

## Procedural Generation

In order to develop a game with automated generation, I will need to make use of certain algorithms. I need to use an algorithm which makes rooms (like dungeons) rather than land (like games such as Minecraft). I could use procedural generation for the shooting level of the game.

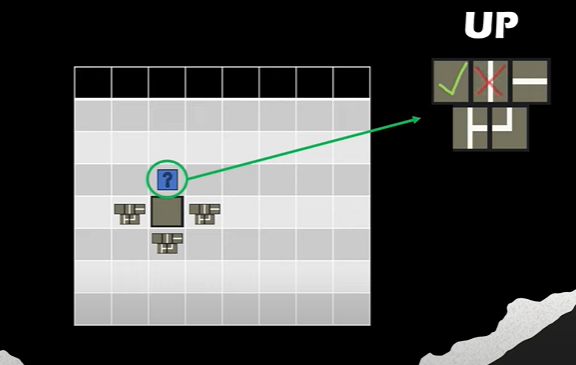
### Binary Space Partitioning



This method, in theory is quite simple to I implement. First you have a plane (A) and then it gets split at a random point creating two rooms. This process is repeated to result in many different sized rooms which can be connected to make a dungeon like map. This is good for my game as I want every level to have a different aim for the players.

### Wave Collapse Function

First you define modules and set up a grid with cells where every tile has an equally likely chance of getting picked per cell. The picks modules based on the adjacent cell.



In this example, the cell which is above the initial cell can have the chance of being a blank cell or a horizontal line. It can’t be a vertical line as it doesn’t fit the tile next to it. Another way to think about this is matching up the edges of the tiles. Using this logic, the tile which is blank is a possible tile to be used but the tile with the vertical line isn’t. This may the best way to code the algorithm. After creating the possible tile combination, the algorithm sees which tile has the least number of possible options and chooses a random option for that cell. The algorithm then updates the possible tiles and loops until all tiles have been chosen.

## Grappling Ability

I will be implementing a grappling ability where the player shoots a grapple towards the point where they want to go and the player gets moved to the point, I will be using a lot of physics here so I will have to either write all my movement scripts to fit a rigid body and character collider or I can switch between a character controller and rigid body when moving and grappling (or anything else physics related). I prefer the latter as my client is new to gaming and using a rigid body to push a player around depending on the input is unnatural compared to using a character controller which is smoother. Also, from my questionnaire, most people ranked accurate physics as the number one component in a game.

A graph with a line and a point

Description automatically generated with medium confidence

Above is a diagram showing how the grappling is going to work. The CalculateJumpVelocity function will require a start and end point and a trajectory height (which is the green arrow in the diagram). This function will return a Vector3 which is the velocity to be added on to the player for it to move towards the grapple point. First, you work out the lowest point of the character (this is why I take away 1 from the y value), then I will work out the difference between this point and the grapple point and I w`ill call this grapplePointRealitveYPos.

Next, I will need a path for the character to take when grappling so that it seems more natural rather than moving straight to the point (I will need to switch to a rigid body here). A constant value called OvershootYAxis will be added on to grapplePointRealitveYPos to indicate the highestPointOnArc (trajectory height). If the player grapples below, then the highestPointOnArc will be the OvershootYAxis.

#### CaculateJumpVelcocity()

A graph of a function

Description automatically generatedTo calculate the jump velocity to push the player I will be using SUVAT equations.

A math equations on a graph paper

Description automatically generatedFirst, I will need to work out a general case and I can implement this into the function. This shows a grapple where A is where the player is, and P is the point where the grapple is. As the trajectory needs to be curved, I will split the situation up into its vertical and horizontal components (indicated by the 3 arrows). PX is the distance between A and P and PY is the distance between the lowest point on A to P. g is the acceleration downwards due to gravity. h is the highest point on the trajectory.

Above is the suvat calculations for the initial arrow. We need to find the initial velocity (horizontal) as well as the time (down). I derived an equation in terms of initial velocity using the suvat equation v2 = u2 + 2as as well as an equation for time using the suvat equation s = vt – ½ (at2).

A math equations on a graph paper

Description automatically generated

Here I am resolving downwards for the last arrow. Here I will not need to work out the time (down), but I will need to find an equation for final velocity. I did this by using the suvat equation s = ut + ½ (at^2).

A math equations on a graph paper

Description automatically generated

Here I am resolving horizontally for the middle arrow where I used the suvat equation s = ut + ½ (at^2) to find initial velocity. Using the time going up and down as well as the initial velocity going horizontally which I worked out before, gave me a general equation for initial velocity.

A math equations on a graph paper

Description automatically generated

A diagram of a graph

Description automatically generatedA diagram of a graph

Description automatically generatedFrom this we can conclude the above triangle’s horizontal and vertical initial velocity equations for the character to grapple from point A to P.

These are Flow charts on the grappling and calculating the velocity. I will split this into at least two functions.

The Rigid body, Box Collider will be active and character controller inactive when Executing Grapple. The opposite is true when Stopping Grapple. (I will have Stop Grapple and Execute Grapple as a separate function).

## Hierarchy Diagrams

### A diagram of a company Description automatically generatedOptions Menu

### Main Menu

A diagram of a company

Description automatically generated

The Main Menu will also have a “Help” which goes to a new scene with a canvas that just explains how all the levels work and “Controls” which is a new scene with all the controls.

## Movement Script

A diagram of a flowchart

Description automatically generatedFor the movement script I will be using Unity’s free look camera as well as the character controller. Here is the initial flow chart for it.

A graph paper with math equations and formulas

Description automatically generatedWhen the player looks in the direction they want to move and press the forward key, the character must rotate in this direction. This is less of an issue if they want to move backwards or side to side as it is a set angle they need to rotate. For the rotation I will be using quaternion.euler() which will generate an orientation based on the input angle provided.

Above is a top-down diagram that shows how you can use the (x,y) coordinates of where the player needs to move to depending on the camera direction and how you can rotate it. The red circle is the character, and the red arrow represents where the camera is facing which is where Unity measures the angle from. The right-angled triangle on the top shows the general case on how you can work out the angle of rotation based on the coordinates.

A cartoon character on a white surface

Description automatically generated

The image above shows the player in the third person camera. The camera is available to move around the character but when the player presses the W key to go forward the player will rotate to go forward. This contrasts to the combat camera (start of the next page) where the player faces the forward direction all the time. This is more consistent for shooting and grappling; the crosshair also helps with aiming in this mode.

A cartoon character on a white floor

Description automatically generated

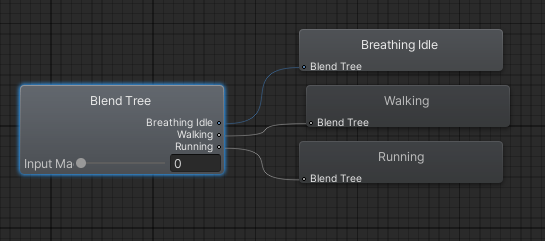
Below is also an animation tree on how the animations are controlled for the player, the base state is breathing idle. When the player moves, the moving blend tree is activated. The blend tree takes in an input magnitude to indicate how fast the animation needs to be for moving. This makes for a smooth transition from walking and sprinting animations. The moving blend tree goes back to idle when the player is not moving.

Blend trees are useful for controller players as they can control the amount of force, they can input using the joystick. Although I don’t plan for controller support for my game, this could be a future feature. If I do want this feature, I will need to implement root motion for characters as this will make moving feel more dynamic (the more you push the joystick the faster the player moves).

From the idle state there are one more state, jumping then goes to falling which then goes to landing and back to the base idle state. The animations will be controlled by the movement script by Boolean variables.

A screenshot of a computer

Description automatically generated



Moving Blend Tree which takes input magnitude and transitions from states 0-1 which is idle to walking. 1-2 is walking to running animation speed.

## Enemies

I’m going to be using the Nav Mesh agent in Unity which will help with moving the player around by splitting the 3D space into many polygons so that it can follow path finding algorithms.

On the next page there is a flow chart of how the enemy will react with the player. The attack range will have a smaller area than the sight range. So that the enemy moves towards the player before attacking.

Below is an image of an enemy. The lines around it indicate the sight range (yellow) and the attack range (red) in 3D space.

A computer generated image of a diagram

Description automatically generated with medium confidence

A diagram of a football game

Description automatically generatedA diagram of a set of positions

Description automatically generatedA diagram of a random algorithm

Description automatically generated with medium confidence A diagram of a game

Description automatically generated

### Nav Mesh Agent

Nav Mesh is a component that helps characters to avoid each other, move around the scene toward a common goal. Once baked into the scene they include areas where the character can walk, avoiding any obstacles.

A drawing of a network

Description automatically generatedNav Mesh use A\* pathfinding which is an extension of the Dijkstra’s Algorithm. It makes a graph with many polygons with the vertices representing nodes. It starts at the start node, and it analyses the graph to find the shortest path between that node and all the other nodes in the graph. The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path. Once the algorithm has found the shortest path between the source node and the other node, that node is marked as visited and the previous node to get to the current node is recorded. The process continues until all the nodes in the graph has been added to the path with the shortest distance to get there. From this it is possible to work out the shortest possible path to reach each node.

Above is a simplified diagram, it shows many nodes A – E with a starting point S and ending point E (green). As it starts at S, the algorithm looks at nodes A and B (as they are the possible paths) in this case B will be the preferred path as 2 is less than 7. A is then marked as visited as all possible paths have been looked at.

Next, we look at B as it isn’t marked as visited and is the shortest path from S. Then we consider each node connected to B (A and D). D is the preferred path with distance 2 + 1 = 3 from S as A has distance of 2+3 = 5 from S. The value of A (via B) is shorter than the distance via S so the shortest distance to get to node A is updated. B is now marked as visited.

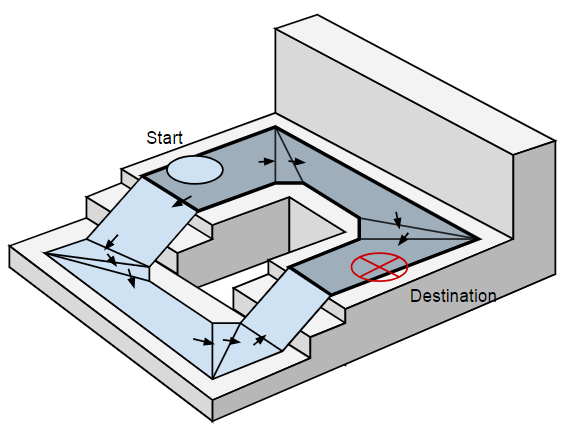
A diagram of a network

Description automatically generatedThis process is repeated until all nodes have been marked as visited. This means that all paths have been explored so we can confirm that all the nodes are marked with the shortest distance to get to them. From this we can see the shortest path to get to E (as long as the previous node has been specified). The disadvantage of this is that if you had many nodes that are all uniform in weight. Dijkstra’s would look at every single path which is in efficient.

A diagram of a square shaped structure

Description automatically generated with medium confidenceA star introduces a new component which contains the physical distance from the start to the target node. This is added to the weight of the path to give a new value, this give preference to nodes that are going roughly in the right direction. This makes it more efficient. From the example above, as B is closer to the end node compared to A, it makes B take preference.

NavMesh creates convex polygons to represent nodes which are a useful representation, as we know that there are no obstructions between any two points inside a polygon. In addition to the polygon boundaries, NavMesh stores information about which polygons are neighbours to each other.



To find path between two locations in the scene, NavMesh maps the start and destination locations to their nearest polygons. The A\* path finding algorithm (explained previously) is used here with the edge weights being the distance of getting to the next polygon and the shortest path is determined.

A close-up of a plane

Description automatically generated

Once a path has been found it will steer the agent to each visible corner of the polygons defined to be the shortest path. For the enemy I will make a random destination within the constraints of the NavMesh. This will resolve the logical error of setting a destination on an obstacle.

A diagram of a diagram of a diagram

Description automatically generated with medium confidence

For collision avoidance of obstacles, the NavMesh agent cuts out a hole in the Mesh depending on if an object is touching the Mesh, this is represented by the blue outline.

## Maze Generation

As a level in my game, I will be implementing a maze generation algorithm to generate random mazes every time to make the game unique every time. Although my client preferred no procedurally generated levels, this will not be too overwhelming for my client, and it would make the game more enjoyable.

A maze with a red dot

Description automatically generatedA grid with a red dot

Description automatically generatedA black and white square with a red dot

Description automatically generatedA black and white square with a red dot

Description automatically generatedA maze with a red circle

Description automatically generated

The above sequence of images shows how the maze generation will work. Initially the maze is blank, the algorithm starts in the top left corner of the grid and moves to a random neighbouring cell that has not yet been visited, breaking the wall in the process. In this example, the algorithm moves right first, moves down twice, left and back up again (where it gets to a dead end). The algorithm will then go back the way it came to find more neighbouring cells to “knock down”. In this case, the algorithm moves down and finds another cell to move into. The process repeats and the algorithm gets stuck again (image 4) and moves back to finish the maze. As the algorithm breaks down the cell using a random number generator, this will likely make different mazes every time. The algorithm will always make a maze which is possible to complete as it visits every cell at least once.

Below is the class diagram for the two scripts maze generation which will implement the algorithm for the generation of the maze. The maze cell script is a child of maze generation, and it represents one maze cell in the grid. This script will contain functions like clearing walks around the maze cell to make a path.

A diagram of a cell

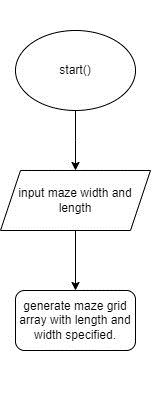
Description automatically generated

A diagram of cell phone

Description automatically generated with medium confidenceBelow is a series of flow chart representing the information. Each flow chart is a function.

A diagram of a cell

Description automatically generated I updated the flow chart with a loop with condition “while next cell is not empty” (explained in testing).



A maze with a green background

Description automatically generatedA diagram of a cell

Description automatically generatedA diagram of a cell

Description automatically generatedMaze Generation flow charts continued.

To the left is how the maze looks in the game

## Inventory

For the inventory of the player, I will be using a database to organise the data. The use of a Scriptable Object architecture will make it so that we don’t have to link to the player as a game object to find its inventory, instead it will be in the Assets folder of the project. This will future proof the game in terms of not needing to reference new players or items that may be added to the inventory script.

### Items

I will be using a base class of Item Object so that I can add other child classes from this parent class (for example a Food Object). Below is a class diagram for this.

A screenshot of a computer

Description automatically generated

In the Item Object class there is another class called Item given the name data, this will serve as a container for basic information about every Item such as a name and Id. This class will contain a constructor (a method that does not get inherited by the children objects and does not return a value) which will give a name and Id to the item which can be easily appended to the database.

The Item Type in the Item Object class is an enumerated type. An enumerated type is a data type which contains a set predefined constant. The use of an Enum can increase run time, make the code more readable and improve type safety as it restricts the possible values that may be passed through.

### Database

As you cannot convert the inventory scriptable object to a JSON file, I will need to make a database that holds items so that we can serialise the scriptable object of our item back into the inventory when loading a saved inventory. Serialisation is the process of transforming objects into a format that can be recognised by unity.

Dictionaries cannot be serialised, so I had to the OnAfterDeserialize() function to create the dictionary.

A screenshot of a computer

Description automatically generatedThe database will be a scriptable object which contains a dictionary with the ids and Item Objects which make part of the game. The code in this scriptable object will hold all the logic of setting up the database and searching the database. Below is a class diagram and screenshot of the database in the unity inspector.

A screenshot of a computer program

Description automatically generated

### User Interface

The User Interface script is a scriptable object. The class Static Interface and Dynamic Interface inherits from this base object. The Static Interface is responsible for the players equipment inventory and the dynamic interface is responsible for the players’ regular inventory. These scripts contain the logic for what happens when the player enters, exits and drags objects in the inventory to different slots.

### Inventory Object

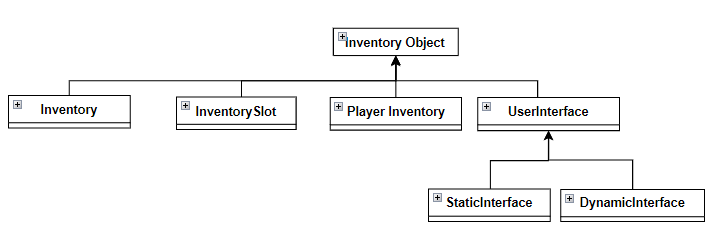
This scriptable object will hold all the logic for the functionality of the player’s inventory and contains methods for adding, swapping and loading items as well as saving and clearing inventory.

Inventory slot is a child of the inventory object class which represents individual slots in the inventory which hold information of the items e.g. Items id and type and methods for updating, removing and adding items to slots.

There is also a property in the inventory slot class which returns an ItemObject and uses the get {} construct which returns the current item in the slot. This is not technically a function, so I didn’t include it in the class diagram.

The inventory class is a child of inventory object which holds an array of inventory slots. It has a clear function for clearing the inventory. This class is serialised into a binary function when saving. Likewise, the class deserializes when loading the inventory back up into an array of inventory slots.

Below represents the parent child relationship between classes. (attributes and functions can be found under the technical solution section)



A screenshot of a computer

Description automatically generated

To the left is a screen shot of what the container for the inventory looks like in the inspector. (The player equipment looks the same as this but with 5 slots and allowed items are specified)

### Billboard

As I intended to use 2D sprites for the items on the floor. This means that I will have to make a billboard script so that the sprites on the floor always face the camera. I plan on doing this by using a late update function as it does not need to happen at the same time as an update function. This can slightly decrease the processing power needed.

### Saving and Loading files

Instead of using Unity’s default way of storing player data by player prefabs it would be a lot more secure by using a file format like JSON or XML for storing data. Although they are easy to use and modify this can be a bad thing as players can change data, so this is not very secure either. Therefore, using a binary formatter to convert a file into binary would be better as they cannot be easily read as they are in binary, so they are more secure.

Items are converted to an Item class which stores in the player’s inventory. This can be saved using a Binary formatter which formats data in a way that it cannot be read (and therefore cannot be edited) by humans.

Below are screen shots of inventories which have been saved by a binary formatter.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Below is a picture of the inventory system in action. The equipment inventory is the one on the left where players can equipment items. You can only have specific items in specific slots e.g. you can only have a helmet on the top, chest plate in the middle and boots at the bottom. You can place balls, keys, bones and grapples in either the left or right slots. Having a ball in the equipment slots means the player can shoot. If you have a grapple in your equipment slots you can grapple. If you have both you can use both grappling and shooting. Also, some items are stackable, the red balls and bones have a number by them indicating the amount e.g. there are 22 red balls and 4 bones (if there is no number it indicates that there is one of that item for example, the helmet and other armour pieces are not stackable).

A screenshot of a video game

Description automatically generated

A screenshot of a video game

Description automatically generatedA screenshot of a video game

Description automatically generated

The pictures above show that only certain items can go into the equipment inventory. Top can only have a helmet. Middle can only have a chest piece. Bottom can only have boots. Left and right can either have keys, bones, balls, or a grapple. (You need to have the balls and grapples equipped in order to use them)

## Login System

A login system will be shown at the start of the game so that users can login and save personal data for the game. I would like it to implement a hash map within the PlayFab servers for faster read and write times when accessing and writing data which is important as the number of entries (players) increases. However, I had trouble trying to implement a hashmap using PlayFab’s API into the PlayFab servers however I will still be making a hashmap for the future of the game.

### PlayFab API

PlayFab is a backend platform for many AAA games such as Minecraft and rainbow six siege. In my game, I will be using PlayFab for storing login details by using its API. Even though PlayFab is a well trusted and secure server, I do not want the possibility of leaking login information. Initially I wanted to use an encryption method to hold these passwords, however a hash may be a better option.

To use PlayFab in unity you must use API requests to updates tables. PlayFab and many other web-based servers use the RESTful API. Initially Unity makes a HTTP request for writing or reading data then the RESTful API translates HTTP requests into SQL requests in the database stored on the PlayFab servers.

### PlayFab

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generatedAbove is an image of the email vzl38162@zbock.com stored in the server. We call the send reset mail command from the script by using PlayFab’s API if the player has forgotten their password and wants to change it.

To the left is an example of an email that has been sent to reset the password of the account vzl38162@zbock.com

A screenshot of a login page

Description automatically generatedA close-up of a white background

Description automatically generated

Above are the reset password pages from Playfab.

A screenshot of a computer

Description automatically generatedAt the end of the game the user will get a score depending on how many bones they collect in the last level of the game. This score is then recorded and sorted into a leaderboard with the value and the PlayFab ID “Name” of the player that achieved this score which can be used to work out the email that got the score.

### Hashing

A diagram of a block diagram

Description automatically generatedHashing is the process of taking a string of any length and outputs a hashed string of a fixed length. This is better than encryption because you cannot undo a hash, but you can undo an encryption by decryption. This makes it impossible to decrypt a hash, instead you must compare the two hash strings to see if they are the same. For my login system I will hash the password to be stored in the PlayFab servers and then compare hashes to the hashed password stored in the server. The hashing algorithm I will be using is shown by the flowchart below. It works by calculating a large decimal value by using the input hash. This will then be turned into a hexadecimal value.



This is the result from the coded solution of the flow chart above. The first hash represents “hello world”, the second hash represents “Hello World”. The hash will be the same length regardless of the intial length of the string however the two hashes are very different even though only two letters have been changed to be a capital.

A diagram of a flowchart

Description automatically generatedAnother use of hashing is creating a HashMap of entries. This takes the entry (for example the password) and produces an integer hash value. This value and password are stored as a key-value pair in a dictionary. By doing this means it is fast to find values in this dictionary as you can hash the value of the password to find the ordinal position in the dictionary of the value. If two passwords produce the same hash value, there will be a collision. In the case of a collision there will be a skip factor of 1 so the password will go into the next available space. The flow chart below represents how the algorithm works.

A screenshot of a computer program

Description automatically generatedIn my hashmap, I will be using a simple modulo (%) 100 calculations. This is sufficient for now as the number of entries will be low, however, as the dictionary gets larger, the entries would become more clustered making a waste of memory and slower searching speeds (as there are more potential collisions). For the future of the game, hash maps may be used to sort data.

A screenshot of a computer

Description automatically generatedAbove is a picture of the HashMap storing names in particular positions in the dictionary, the name “johnsmith” and “smithjohn” will have the same hash value as they contain the same letters. This causes a collision with the value “johnsmith” with “smithjohn” at position 80 and “johnsmith” gets put in position 81.

Debug.Log(getKey("johanjobi"));

Debug.Log(getKey("smithjohn"));

Debug.Log(getKey("johnsmith"));

Debug.Log(getKey("jobijohan"));

The getKey function takes into account the collision and looks at the next position in the dictionary for the value. This is why it is available to find johnsmith in position 81. The value “jobijohan” is not in the dictionary therefore the value of -1 is returned to indicate that it is not in the dictionary.

# Testing and Debugging

## Movement Script

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | WASD buttons inputted | Character moves in the respective direction and walking animation plays | P when moving in diagonal speed increases |
| 2 | Shift key inputted whilst WASD is being inputted. | Character moves faster in the direction and running animation plays | P |
| 3 | Mouse aim inputted | Camera pivots around player | P but quite snappy when inputting movement after |
| 4 | No movement keys inputted | Idle animation plays | P |
| 5 | Spacebar Input | Jump animation plays force and acts on character to propel upwards | P |
| 6 | Player presses C | The camera toggles between normal camera mode and combat camera mode which has a crosshair | P |
| 7 | Player falls from a height | Falling animation plays and gravity propels the character downwards | P |
| 8 | Player looks around using the camera in normal camera mode | The character stays still and the camera pans around the player | P |
| 9 | In Normal camera mode, Player moves camera to look at a different direction to where character is facing . Player then attempts to walk forward | The character rotates to the direction the camera is facing and walks forwards in that direction | P |
| 10 | Player looks around using the camera in combat camera mode | The character rotates as the player looks around. Crosshair always stays in the middle | P |

One of the problems of the movement script was, when changing direction, the character would snap to the direction that the player was moving in. I wanted smooth gameplay, so I used the Mathf.SmoothDampAngle command.

Another problem was, when moving in a diagonal direction, the speed was increased. This was because in a triangle the hypotenuse (which is the diagonal), is the longest side and therefore will have a larger value than the forward and back components (by Pythagoras). To fix this, I normalized the vector using .normalized. This means that the vector is divided by its magnitude to get a unit vector in the direction of movement.

I initially used an animation tree for walking, running and idle. However, this was not that smooth as when the player wants to run the character would snap into the running animation. This is important as I want the game to be as realistic as possible. To combat this, I used a blend tree for the idle, walk and running phase which takes an input magnitude that can be used to transition between the states. When the input magnitude is between 0 and 1, the character is in between an idle and walking state and between 1 and 2, the input magnitude is between a walking and running state.

## Throwing Behaviour

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail +comments |
| 1 | Left Mouse button pressed | Projectile is shot accurately by crosshair | F slightly left of crosshair |
| 2 | Player shoots projectile and hits Enemy | Enemy takes damage per shot | F damage is continuously registered |
| 3 | Enemy Shoots projectile and hits player | Player takes damage | F projectile gets stuck on enemy and enemy takes damage |
| 4 | Projectile Hits anything other than player or enemy | Projectile game object gets destroyed | F shots that enemy shoots do not get destroyed |
| 5 | Projectile does not hit anything | Projectile gets destroyed | F |
| 6 | Shooting whilst not in combat mode | Player should not be able to shoot | P |
| 7 | Throws runout | Player should not be able to shoot anymore | P |

When shooting the projectile goes slightly to the left of the crosshair. This is because I was instantiating an object in front of my player and, because I am using a 3rd person camera, it is not on the crosshair exactly. To fix this I used a ray cast from the middle of the screen for shooting.

When a singular shot is on target, damage is taken all the way down to -1. This happened rapidly so I knew this was because an if statement was running in void Update (or similar). The collision enter function is continuous so, the damage is taken multiple times if the game object touches for multiple frames. I have instead used the OnTriggerEnter() function so that it only registers one unit of damage when the projectile game object hits and gets destroyed.

Another problem was every time the enemy shoots a projectile, it takes damage as it touches the collider. To fix this problem, I created a new projectile and script for the enemy.

If a projectile gets shot into the air, it keeps travelling up until the game stops. This takes up unnecessary amounts of memory therefore potentially slowing the game down. So now the projectile deletes itself if it has been instantiated for longer than 5 seconds.

## Grappling

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail +comments |
| 1 | Right click pressed | Visual grapple spawns in line with cross hair | P |
| 2 | Character moves when grappling | Player movement is stopped only force acting is by grapple | F player can still move when in the air whilst grappling |
| 3 | Collision after grapple | Player should be able to move normally | F player is still getting pulled to the grapple point |
| 4 | Grapple on object too far | Grapple visually shows that the distance is too far | P |
| 5 | Grapple on object lower than player | Player moves towards grapple point with a small jump | P |
| 6 | Grapple to a point above player | Player should initially get propelled over the grapple point but then loop down and make it to the grapple point exactly | P |
| 7 | Grappling whilst not in combat mode | The player should not be able to grapple | P |

After grappling the player is not able to move and a force in the direction of grapple is still acting on the player. This means that the set velocity function is not being stopped.

To fix test 2 I can freeze the character controller and make any velocities before shooting the grapple 0. The best way to do this is by using rb.velocity = Vector3.zero;as the character controller will be disabled during grappling.

I have used the is kinematic option on the rigid body and I disabled the box collider on the player to stop it from taking an effect on the controller during regular movement.

## Maze Generation

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | Maze algorithm starts | Maze follows algorithm (mentioned in documented design). Maze is different each time. | F the maze stops in halfway through |
| 2 | Maze dimensions | The player should be able to fit through the maze | F the maze is too small |
| 3 | Maze completion | Every cell is visited | F it only generates a path from start to finish and doesn’t visit every cell |

The original maze is too small. I used a parent object for the maze so that when scaling up everything is proportionate. However, when doing this, all the maze cells were overlapping with each other. This was because “Instantiate” uses world positions rather than positions local to the parent object.

Algorithm would not go to every cell in the maze, instead it would stop until it reaches a cell with no unvisited neighbours. Therefore, I made a loop which generates a maze until the cell has neighbours which have all been visited, but then goes back to the previous cell to check for any unvisited neighbours. This way the algorithm would have looked at every cell in the grid.

## Enemy

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | Player and Enemy get close to each other | Enemy moves to player and starts attacking | P after attacking the enemy keeps attacking the player even if the player moves away. Also, enemy gets too close to the player |
| 2 | Player in sight range but not in attack range | Enemy moves towards the player | P |
| 3 | Player in attack range | Enemy shoots at player | P but the enemy is not looking at the player it is shooting at |
| 4 | Enemy’s health runs out | Enemy object gets destroyed and items are dropped | P |

For test 1, I have made an alreadyAttacked variable which is set to true when the player has already been attacked and turns to false when the player has not been attacked (in the shooting cooldown of the enemy) this makes a buffer between attacking and not attacking so that the player can escape from the enemy. For the chase player function I have used the players position as the coordinates that the enemy moves to. When the player starts attacking (player is in attacking zone) I have made it so that the enemy stops moving when it is attacking so that the enemy does not go right up to the player.

I have added transform.LookAt(player) so that the player object rotates to the direction of the player.

## Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | P key is pressed when in the game | Pause menu shows and game freezes, all buttons function correctly | P |
| 2 | Game starts | Player greeted with starting menu, all buttons function correctly | P |
| 3 | Resolution is changed | Finds all available resolutions and displays them. Switches depending on which one the user chooses | P |
| 4 | Game Graphics changed | Game graphics render more/less depending on the choice | P |
| 5 | Quit button pressed | Game stops | P |
| 6 | Volume is changed | Volume of the system changes | P |
| 7 | Full screen is toggled | Toggle between windowed and floating window | P |

## Inventory

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected result | Pass/Fail + comments |
| 1 | Items placed in player equipment inventory | Items swap if they are allowed to be in that slot | F most of the time this is a P however sometimes it creates a sprite that does not go into the inventory slot |
| 2 | Items picked up by player | All Items should have the unique IDs in the database so should not clash | F there are duplicate keys in the database |
| 3 | User attempts to save and load an inventory | The saved items should appear in the inventory and equipment | F there was a unity error when loading |
| 4 | User swaps Items and moves them around slots | The item should follow the mouse cursor and move towards the respective slot. This should be logically updated as well as graphically updated | P however it takes up a lot of processing power and sometimes crashes the game because this is under the update function which is called every frame |
| 5 | Player walks over an item on the ground | The item should go into the inventory of the player | P |
| 6 | Grapple moved to equipment | Grapple should not work unless it is in the equipment slot | P |
| 7 | Red Balls moved to equipment | Player should not be able to shoot red balls unless it is in the equipment slot. | P The number of red balls in the equipment is the amount they can shoot |
| 8 | Armour in equipment slot | The player should gain 20 HP for every piece of armour they have on | P |
| 9 | Damage taken while armour is on the equipment slots. | The damage should be taken off the armour health before the players health. When the armour health gets to 0, the armour is destroyed | P |
| 10 | Food is placed in the equipment slots | Player health regenerates depending on the amount of food there is. The food item gets deleted from the equipment after it has regenerated the players health. | P |
| 11 | Items dragged out of inventory | The item will get deleted | P if you miss a slot on the interface when dragging it will delete the item |

Test 1 was hard to reproduce and therefore difficult to debug. The bug happened every time after the user destroys an item via the drag and drop outside of the equipment and inventory panels but then attempt to drag, from inventory to the slot that item that just got destroyed was in.

Test 2 I used an on before serialisation method which clears the dictionary every time before the game serialises in unity and then auto populates the dictionary again.

For test 3, The code for saving was attempting to save a scriptable object (User Interface) which you can’t do as it is like trying to save a game object. Instead, you will need a reference to the object, so I used [System.nonSerialisable] on the User Interface variable called parent in the inventory slot class.

For test 4, my code used for loops which looped through every slot in the player inventory to check if it has been change by the user swapping or removing the item. This is quite processing heavy. Instead, there is now a callback using a delegate.

A delegate is a variable that holds one or more methods, the significance about them is that they can use event listeners. This makes them ideal for an event driven systems which is what I will need so every time an item undergoes a change, it will update only that one item. This will make it less processor heavy. Since doing this, the slots are stored inside the inventory database container which would causes more memory usage however I think it is a good trade-off.

For test 11 Another problem with the system was every time the player dragged an item from the inventory and misses the slot it will get deleted. Instead, I made a variable called item over interface which checks if the item is over either the equipment or inventory canvas borders. When the player misses the slot, but the item is hovering over the canvas borders, it won’t delete the item and instead revert the action.

Previously I had a dictionary which returns an ID when you input an Item Object, if you wanted to get and Item Object, it would loop through every item in the dictionary and find the Id which matches the one requested. This was very processor heavy and with performance in mind I decided to make two dictionaries instead. This is more memory heavy, but it is a good trade off as at the moment, there is not a lot of items in my game. In the future, if there are many more items, this may change.

public Dictionary<ItemObject, int> GetId = new Dictionary<ItemObject, int>();

public Dictionary<int, ItemObject> GetItem = new Dictionary<int, ItemObject>();

## Login System

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected result | Pass/Fail + comments |
| 1 | Login pressed with invalid login email or password | “Incorrect email or password” displayed on screen | P |
| 2 | Register pressed with email already in use | “Email already in use” displayed on screen | P |
| 3 | Reset password pressed with no email | “Enter email address” displayed on screen | P |
| 4 | Reset password pressed with email | “Password reset email sent” displayed on screen and password reset email is sent by and records are updated | P |
| 5 | Reset password pressed with email but user does not change password | “Password reset email sent” displayed on screen and password reset email is sent but records are not updates | P |
| 6 | Player registers with a valid email and password | Updates PlayFab servers with username and hashed password and they go straight to the main menu of the game | P |
| 7 | Player logs in with valid password and email | email checked against records in PlayFab servers. Passwords is hashed then checked with the hashed password stored in PlayFab servers when the user first registers. | P |
| 8 | Player gets score X at the end of the game | Score X is stored in the PlayFab servers. A leaderboard is created depending on which account has the highest score X. | P |
| 9 | Reset password button pressed with email that has not been registered on the PlayFab server. | “Invalid email address” outputted | P |
| 10 | User registers with an email that does not meet the password requirements | “Password too short (min 6 characters)” displayed and user has to enter a new password. | P |

## HashMap

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | New entry is added | When adding a new entry, a hash is calculated and is added to the dictionary at the location calculated | P |
| 2 | Location of entry is searched for | Hash is calculated and looked up in dictionary | P |
| 3 | There is a collision when adding data | Increases the hash by the skip value and puts it in that location. It will keep increasing the hash by the skip value until it finds a free space | P |
| 4 | Location of entry is searched for, but value is not found | Algorithm returns -1 | P |

## Hashing Algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test Description | Expected Result | Pass/Fail + comments |
| 1 | Value is passed through | Hexadecimal value is calculated and returned | P |

Pictures of the HashMap and Hashing Algorithm working can be found in the documented design section.

# Technical Solution

## A screenshot of a computer Description automatically generatedMovement Script

using JetBrains.Annotations;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Net;

using System.Runtime.CompilerServices;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.ProBuilder.MeshOperations;

public class MovementScript : MonoBehaviour

{

[Header("Refrences")]

public CharacterController controller;

public Transform cam;

public Animator animator;

public Transform groundCheck;

public Rigidbody rb;

public BoxCollider Boxcollider;

[Header("Slope Handling")]

public float maxSlopeAngle;

private RaycastHit slopeHit;

private bool existingSlope;

[Header("Checks")]

public float groundDistance = 0.4f;

public LayerMask groundMask;

[Header("Input")]

bool isGrounded;

bool isJumping;

[Header("Jump")]

public float jumpButtonGracePeriod;

private float? lastGroundedTime;

private float? jumpButtonPressedTime;

public float jumpHeight = 3f;

[Header("Physics")]

public float maxSpeed = 6f;

public float turnSmoothTime = 0.1f;

public float gravity = -9.8f;

float turnSmoothVelocity;

Vector3 velocity;

[Header("Grappling")]

Vector3 moveDirection;

public bool activeGrapple;

public bool freeze;

private Vector3 VelocityToSet;

private bool enableMovemenentOnNextTouch;

private void Start()

{

Cursor.visible = false;

Cursor.lockState = CursorLockMode.Locked;

animator = GetComponent<Animator>();

rb = GetComponent<Rigidbody>();

Boxcollider = GetComponent<BoxCollider>();

controller.enabled = true; // box collider and rigidbody disabled for reguluar movement

Boxcollider.enabled = false;

rb.isKinematic = true;

}

void Update()

{

Movement(); // for future development of the code I have used the void update to call the Movement Function

}

private void Movement()

{

if (activeGrapple) return;

// if active grapple then the player doesnt move

//if (freeze) velocity = Vector3.zero;

// make the velocity 0 for grappling so you dont continue to move (you freeze)

if (freeze)

{

rb.velocity = Vector3.zero;

maxSpeed = 0f;

}

else

{

maxSpeed = 6f;

}

isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

if (isGrounded && velocity.y < 0)

{

velocity.y = -2f; // implemets gravity when grounded

}

float horizontal = Input.GetAxisRaw("Horizontal"); //gets horizontal and vertical input from mouse

float vertical = Input.GetAxisRaw("Vertical");

Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;// direction which you are looking, .normalized

// takes the vector of the same direction and makes the length 1

float inputMagnitude = Mathf.Clamp01(direction.magnitude); //clamps magnitude between 0 and 1 for value in animator blend

//tree

if (Input.GetKey(KeyCode.LeftShift))

{

inputMagnitude \*= 2; //increases speed of animation when running

}

animator.SetFloat("Input Magnitude", inputMagnitude, 0.05f, Time.deltaTime);

float speed = inputMagnitude \* maxSpeed;

// this is useful as without it, moving in a diagonal will make it faster

if (direction.magnitude >= 0.1f)

{

animator.SetBool("isMoving", true);//for animation

float targetAngle = Mathf.Atan2(direction.x, direction.z) \* Mathf.Rad2Deg + cam.eulerAngles.y;

// finds target rotation for the character and adds current y value of camera to determine where the character should face

float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y, targetAngle, ref turnSmoothVelocity, turnSmoothTime);

//smooths the rotation angle towards target angle to make it more smoother

transform.rotation = Quaternion.Euler(0f, angle, 0f);

Vector3 moveDir = Quaternion.Euler(0f, targetAngle, 0f) \* Vector3.forward;

// moves direction of character when you start moving and change the mouse direction

controller.Move(moveDir.normalized \* speed \* Time.deltaTime);

//moves the character at a consistent speed regardless of the input direction magnitude the time.deltatime

//makes for smooth movement independant of the fram rate

}

else

{

animator.SetBool("isMoving", false);

}

if (isGrounded)

{

lastGroundedTime = Time.time; //find time of last grounded time

}

if (Input.GetButton("Jump"))

{

jumpButtonPressedTime = Time.time; //find the time since jump pressed

}

if (Time.time - jumpButtonPressedTime <= jumpButtonGracePeriod && Time.time - lastGroundedTime <= jumpButtonGracePeriod)

{// this is used to make the jump more responsive and uses a grace period so that you dont have to press the jump

// exactly at the frame where the player is grounded making parkour a bit easier for the player

velocity.y = Mathf.Sqrt(jumpHeight \* -2f \* gravity);

animator.SetBool("isJumping", true);

isJumping = true;

jumpButtonPressedTime = null;

lastGroundedTime = null;

}

//gravity

velocity.y += gravity \* Time.deltaTime;

controller.Move(velocity \* Time.deltaTime);

if (Time.time - lastGroundedTime <= jumpButtonGracePeriod)

{//animations

animator.SetBool("isGrounded", true);

isGrounded = true;

animator.SetBool("isJumping", false);

isJumping = false;

animator.SetBool("isFalling", false);

if (Time.time - jumpButtonPressedTime <= jumpButtonGracePeriod)

{

animator.SetBool("isJumping", true);

isJumping = true;

jumpButtonPressedTime = null;

lastGroundedTime = null;

}

}

else

{

animator.SetBool("isGrounded", false);

isGrounded = false;

if ((isJumping && velocity.y < 0) || velocity.y < -2)

{

animator.SetBool("isFalling", true);

}

}

}

private Vector3 CalculateJumpVelocity(Vector3 startPoint, Vector3 endPoint, float trajectoryHeight)

{

float gravity = Physics.gravity.y;

float displacementY = endPoint.y - startPoint.y;

Vector3 displacementXZ = new Vector3(endPoint.x - startPoint.x, 0f, endPoint.z - startPoint.z);

Vector3 velocityY = Vector3.up \* Mathf.Sqrt(-2 \* gravity \* trajectoryHeight);

Vector3 velocityXZ = displacementXZ / (Mathf.Sqrt(-2 \* trajectoryHeight / gravity)

+ Mathf.Sqrt (2\* (displacementY - trajectoryHeight) / gravity));

return velocityXZ + velocityY; // calculates jump velocity returns the y and x velocities

}

public void JumpToPosition(Vector3 targetPosition, float trajectoryHeight)

{

activeGrapple = true;

VelocityToSet = CalculateJumpVelocity(transform.position, targetPosition, trajectoryHeight);

Invoke(nameof(SetVelocity), 0.1f);

// delay in time so that it doesnt apply when the movement control function is still active

}

private void SetVelocity()

{

controller.enabled = false;

Boxcollider.enabled = true;

rb.isKinematic = false;

// box collider and rigidbody enabled for giving a force to the character

enableMovemenentOnNextTouch = true;

rb.velocity = VelocityToSet;

// to push the player into the direction of grapple

}

private void OnCollisionEnter(Collision collision)

{

if (enableMovemenentOnNextTouch)

{

enableMovemenentOnNextTouch = false;

activeGrapple = false;

controller.enabled = true;

Boxcollider.enabled = false; // box collider and rigidbody enabled for grappling

rb.isKinematic = true;

GetComponent<Grappling>().StopGrapple();

}

}

}

## Background Music

backgroundMusic script controls the background music.

A screenshot of a music

Description automatically generated

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class backgroundMusic : MonoBehaviour

{

public static backgroundMusic instance;

public AudioSource music;

void Awake()

{

instance = this;

DontDestroyOnLoad(gameObject); //so the music doesn’t stop playing when switching between levels

}

void Start()

{

// Play the background music when the game starts

PlayBackgroundMusic();

}

public void PlayBackgroundMusic()

{

music.Play();

}

}

## A screenshot of a computer screen Description automatically generatedBillboard

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Billboard : MonoBehaviour

{

public Camera \_camera;

private void Start()

{

\_camera = Camera.main; //separate variable holding camera.main this was previously in void update which took up a of unnecessary processing power to find Camera.main every frame

}

private void LateUpdate()

{

transform.forward= \_camera.transform.forward;

}

}

## A screenshot of a computer program Description automatically generatedButton

(Button script for when a player steps on a button and it activates a parkour course)

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Button : MonoBehaviour

{

public GameObject parkour;

private bool activated;

private void Start()

{

parkour.SetActive(false);//parkour initially deactivated

}

private void OnTriggerEnter(Collider collision)

{

if (collision.CompareTag("Player"))

{

parkour.SetActive(true); //if the collision is a player then the parkour is set to active

}

}

private void OnTriggerExit(Collider other)

{

//when player exits the button, it will disable the parkour course

if (other.CompareTag("Player"))

{

parkour.SetActive(false);

}

}

}

## A screenshot of a computer Description automatically generatedCombat Camera

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CombatCamera : MonoBehaviour

{

public Cinemachine.AxisState xAxis, yAxis;

[SerializeField] Transform camFollowPos;//serialized field so can be seen in editor

private void Update()

{

xAxis.Update(Time.deltaTime); //time.delta time means the frame updates are consistent and therefore doesnt move faster if you are getting more frames

yAxis.Update(Time.deltaTime);

}

private void LateUpdate()

{

camFollowPos.localEulerAngles = new Vector3(yAxis.Value, camFollowPos.localEulerAngles.y, camFollowPos.localEulerAngles.z);

transform.eulerAngles = new Vector3(transform.eulerAngles.x, xAxis.Value, transform.eulerAngles.z);

}

}

## Combat Cam Toggle Button

A screenshot of a computer

Description automatically generated

using Cinemachine;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CombatCamToggleButton : MonoBehaviour

{

private Switching switching;

public MonoBehaviour combatcamscript;

public MonoBehaviour throwingscript;

public MonoBehaviour grapplingscript;

public CinemachineFreeLook firstCamera;

public CinemachineVirtualCamera secondCamera;

public bool CombatCamActive = true;

void Start()

{

// Ensure both cameras are initially enabled or disabled as needed

if (firstCamera != null)

firstCamera.enabled = false;

if (secondCamera != null)

secondCamera.enabled = true;

}

void Update()

{

if (Input.GetKeyDown(KeyCode.C))

{//toggles between cinemachine’s freelook and virtual cameras

ToggleCameras();

}

}

public void ToggleCameras()

{

CombatCamActive = !CombatCamActive;

//disables the script which cannot be used in third person camera mode

firstCamera.enabled = !firstCamera.enabled;

combatcamscript.enabled = !combatcamscript.enabled;

throwingscript.enabled = !throwingscript.enabled;

grapplingscript.enabled = !grapplingscript.enabled;

secondCamera.enabled = !secondCamera.enabled;

}

}

## Default Object

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Default Object", menuName = "Inventory System/Items/Default")]

public class DefaultObject : ItemObject

{

public void Awake()

{

type = ItemType.Default;

}

//for creation of a default item

}

## A screenshot of a computer screen Description automatically generatedDisplay Score

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using PlayFab;

using TMPro;

public class displayScore : MonoBehaviour

{

public InventoryObject inventory1;

public InventoryObject inventory2;

public Item bones;

public int score;

public PlayfabManager manager;

public TextMeshProUGUI text;

void Start()

{

//displays the score which is the amount of bones the players get

bones.Id = 0;

score = inventory1.AmountOfItem(bones) + inventory2.AmountOfItem(bones);

text.text = score.ToString();

manager.SendLeaderboard(score); //Sends the score from the game to the leaderboard in the PlayFab servers using the function in the PlayFabManager script.

}

}

## Dynamic Interface

A screenshot of a computer program

Description automatically generated

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.EventSystems;

public class DynamicInterface : UserInterface //this is attached to the inventory screen so needs the same logic from user interface and needs to inherit some

methods

{

public int X\_SPACE\_BETWEEN\_ITEM; // inventory graphical properties

public int Y\_SPACE\_BETWEEN\_ITEM;

public int X\_START;

public int Y\_START;

public int NUMBER\_OF\_COLUMN;

public GameObject inventoryPrefab;

public override void CreateSlots() //override makes a new implementation of the method inherited from the base class

{

slotsOnInterface = new Dictionary<GameObject, InventorySlot>(); // contains information on which gameobject is in which inventory slot

for (int i = 0; i < inventory.Container.Slots.Length; i++)

{

//iterating through slots in inventory

var obj = Instantiate(inventoryPrefab, Vector3.zero, Quaternion.identity, transform);

obj.GetComponent<RectTransform>().localPosition = GetPosition(i);

AddEvent(obj, EventTriggerType.PointerEnter, delegate { OnEnter(obj); });//called when entering slot

AddEvent(obj, EventTriggerType.PointerExit, delegate { OnExit(obj); });//called when exiting slot

AddEvent(obj, EventTriggerType.BeginDrag, delegate { OnDragStart(obj); });//called when starting to drag item from slot

AddEvent(obj, EventTriggerType.EndDrag, delegate { OnDragEnd(obj); });//called at the end of the drag

AddEvent(obj, EventTriggerType.Drag, delegate { OnDrag(obj); }); //called when dragging object

inventory.GetSlots[i].slotDisplay = obj; //updates display for inventory slot

slotsOnInterface.Add(obj, inventory.GetSlots[i]); // adds the slot and its inventory slot to the dictionary

}

}

private Vector3 GetPosition(int i)

{

return new Vector3(X\_START + X\_SPACE\_BETWEEN\_ITEM \* (i % NUMBER\_OF\_COLUMN), Y\_START + (-Y\_SPACE\_BETWEEN\_ITEM \* (i / NUMBER\_OF\_COLUMN)), 0f);

//calculates position of slots

}

}

## Enemy 1

A screenshot of a computer

Description automatically generated

using JetBrains.Annotations;

using System;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.AI;

using UnityEngine.UIElements;

public class Enemy1 : MonoBehaviour

{

public NavMeshAgent agent;

public Transform player;

public Transform player2;

public GroundItem objectToDropPrefab;

public ItemDatabaseObject database;

public LayerMask Ground, Player, Player2;

public int health = 100;

//patrolling

public Vector3 walkPoint;

bool walkPointSet;

public float walkPointRange;

//attacking

public float timeBetweenAttacks;

bool alreadyAttacked;

public GameObject projectile;

//states

public float sightRange, attackRange;

public bool playerInSightRange, playerInAttackRange;

public bool player2InSightRange, player2InAttackRange;

public healthBar healthBar;

private void Awake()

{

player = GameObject.Find("player1").transform;

player2 = GameObject.Find("player2").transform;

agent = GetComponent<NavMeshAgent>();

healthBar = GetComponentInChildren<healthBar>();

healthBar.setMaxHealth(health);

}

private void Update()

{

//check for sight and attack range of both players

playerInSightRange = Physics.CheckSphere(transform.position, sightRange, Player);

playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, Player);

player2InSightRange = Physics.CheckSphere(transform.position, sightRange, Player2);

player2InAttackRange = Physics.CheckSphere(transform.position, attackRange, Player2);

//logic for the enemy to follow

if (!playerInSightRange && !playerInAttackRange) Patrolling();

else if(!player2InSightRange && !player2InAttackRange) Patrolling();

if (playerInSightRange && !playerInAttackRange) ChasePlayer(player);

else if (player2InSightRange && !player2InAttackRange) ChasePlayer(player2);

if (playerInAttackRange && playerInSightRange) AttackPlayer(player);

else if (player2InAttackRange && player2InSightRange) AttackPlayer(player2);

}

private void Patrolling()

{//logic for enemy to patrol

if (!walkPointSet) SearchWalkPoint();

if (walkPointSet)

agent.SetDestination(walkPoint);

Vector3 distanceToWalkPoint = transform.position - walkPoint;

//walkpoint reached

if (distanceToWalkPoint.magnitude < 1f)

walkPointSet = false;

}

private void SearchWalkPoint()

{

//Calculates random point in range for patrolling

float randomZ = UnityEngine.Random.Range(-walkPointRange, walkPointRange);

float randomX = UnityEngine.Random.Range(-walkPointRange, walkPointRange);

walkPoint = new Vector3(transform.position.x + randomX, transform.position.y, transform.position.z + randomZ);

if (Physics.Raycast(walkPoint, -transform.up, 2f, Ground))

{

walkPointSet = true;

}

}

private void ChasePlayer(Transform focusPlayer)

{/logic for enemy to chase player.

agent.SetDestination(focusPlayer.position);

}

private void AttackPlayer(Transform focusPlayer)

{/logic for enemy to attack player.

//make sure enemy doesnt move

agent.SetDestination(transform.position);

transform.LookAt(focusPlayer);

if (!alreadyAttacked)

{

//Attack Code

Rigidbody rb = Instantiate(projectile, transform.position, Quaternion.identity).GetComponent<Rigidbody>();

rb.AddForce(transform.forward \* 32f, ForceMode.Impulse);

rb.AddForce(transform.up \* 5f, ForceMode.Impulse);

alreadyAttacked = true;

Invoke(nameof(ResetAttack), timeBetweenAttacks);

}

}

private void ResetAttack()

{

alreadyAttacked = false;

}

public void TakeDamage(int damage)

{

health -= damage;

healthBar.setHealth(health);

if (health <= 0) Invoke(nameof(DestroyEnemy), 0.5f);

}

private void DestroyEnemy()

{

Destroy(gameObject);

objectToDropPrefab.item.data.Id = 0;

Instantiate(objectToDropPrefab, gameObject.transform.position, Quaternion.identity); // Instantiates bones to drop when you kill the enemy. (Item Id = 0 for bones)

var RandomId = UnityEngine.Random.Range(0, 8);

ItemObject item = database.GetItem[RandomId];

objectToDropPrefab.item = item;

Instantiate(objectToDropPrefab, gameObject.transform.position, Quaternion.identity); // Instantiates a random item to drop so when the enemy dies it will drop bones and one other random item.

}

private void OnDrawGizmosSelected()

{

//visual representation of the attack and sight range of the enemy in the unity editor

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, attackRange);

Gizmos.color = Color.yellow;

Gizmos.DrawWireSphere(transform.position, sightRange);

}

}

## Equipment object

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Equipment Object", menuName = "Inventory System/Items/Equipment")]

public class EquipmentObject : ItemObject

{

public float attackDamage; // all equipment objects will inherit these properties from this base class

public float defenceValue;

}

## Food Object

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

[CreateAssetMenu(fileName = "New Food Object", menuName = "Inventory System/Items/Food")]

public class FoodObject : ItemObject

{

public int restoreHealthValue;//all food objects inherit these properties.

public void Awake()

{

type = ItemType.Food;

}

}

## Grappling

A screenshot of a computer

Description automatically generated

using System.Collections;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using System.Xml.Serialization;

using UnityEngine;

using UnityEngine.Apple.ReplayKit;

using UnityEngine.EventSystems;

public class Grappling : MonoBehaviour

{

[Header("References")]

private MovementScript pm;

public Transform cam;

public Transform gunTip;

public LayerMask whatisGrappleable;

public LineRenderer lr;

public Item grapple;

private PlayerInventory playerInventory;

[Header("Grappling")]

public float maxGrappleDistance;

public float grappleDelayTime;

private bool grappling;

private Vector3 grapplePoint;

public float overshootYAxis;

[Header("Cooldown")]

public float grapplingCd;

private float grapplingCdTimer;

[Header("Input")]

public KeyCode grappleKey = KeyCode.Mouse1;

private void Start()

{

pm = GetComponent<MovementScript>();

playerInventory = GetComponent<PlayerInventory>();

grapple.Id = 4;//defines the grapple item

}

private void StartGrapple()

{

grappling = true;

pm.freeze = true;

RaycastHit hit;

if (Physics.Raycast(cam.position, cam.forward, out hit, maxGrappleDistance, whatisGrappleable))

{//starts grapple

grapplePoint = hit.point;

Invoke(nameof(ExecuteGrapple), grappleDelayTime);

}

else

{//cancels grapple (object not in range to grapple)

grapplePoint = cam.position + cam.forward \* maxGrappleDistance;

Invoke(nameof(StopGrapple), grappleDelayTime);

}

lr.enabled = true;

lr.SetPosition(1, grapplePoint);

}

private void ExecuteGrapple()

{

pm.freeze = false; //freezes the movement of the player so they can’t move in the air

Vector3 lowestPoint = new Vector3(transform.position.x, transform.position.y - 1f, transform.position.z);

float grapplePointRelativeYPos = grapplePoint.y - lowestPoint.y;

float highestPointOnArc = grapplePointRelativeYPos + overshootYAxis;

if (grapplePointRelativeYPos < 0) highestPointOnArc = overshootYAxis; //if grapple is below the player

pm.JumpToPosition(grapplePoint, highestPointOnArc);

}

public void StopGrapple()

{

pm.freeze = false;//player able to move again

grappling = false;

grapplingCdTimer = grapplingCd;// cooldown starts

lr.enabled = false;

}

private void Update()

{

if (Input.GetKeyDown(grappleKey) && playerInventory.InEquipment(grapple)) //if there is a grapple in inventory

{

StartGrapple();

}

if (grapplingCdTimer>0)

{

grapplingCdTimer -= Time.deltaTime;//cooldown gets less per frame that the player is not grappling

}

}

private void LateUpdate()

{

if(grappling)

{

lr.SetPosition(0, gunTip.position);//grapple visuals

}

}

}

## Ground Item

using System.Collections;

using System.Collections.Generic;

using UnityEditor;

using UnityEngine;

public class GroundItem : MonoBehaviour, ISerializationCallbackReceiver

{

public ItemObject item;

public void OnAfterDeserialize()

{

}

public void OnBeforeSerialize()

{

// previews item sprite in editor and not when building the game otherwise there will be an error when building the game.

#if UNITY\_EDITOR

GetComponentInChildren<SpriteRenderer>().sprite = item.uiDisplay;

EditorUtility.SetDirty(GetComponentInChildren<SpriteRenderer>());

#endif

}

}

## A screenshot of a computer program Description automatically generatedHashmap

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System.IO;

using TMPro;

using System.Runtime.Serialization.Formatters.Binary;

using UnityEditor;

using UnityEngine.SceneManagement;

using System.Xml;

public class Hashmap : MonoBehaviour

{

Dictionary<int, string> hashMap = new Dictionary<int, string>();

public TMP\_InputField email;

public TMP\_InputField password;

public PlayfabManager manager;

public string savePath = "Assets/hashMap.json";

public Hashmap()

{

hashMap = new Dictionary<int, string>(); //defines new hashmap

}

void Start()

{

Dictionary<int, string> hashMap = new Dictionary<int, string>();

if (File.Exists(savePath))

{

LoadDictionary(); //saves dictionary when the hashmap needs to be used again

}

}

public void Register()

{

Add(email.text);//adds email and password to hashmap when a new player registers

Add(manager.HashingAlg(password.text)); //password is stored using a hash

SaveDictionary();

//when a player registers it is held in a dictionary which uses a hashing algorithm to calculate the position in the dictionary

}

void OnDestroy()

{

// Save the dictionary to file when the game exits

SaveDictionary();

}

void SaveDictionary()

{//saves dictionary by a binary formatter

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Create(savePath);

bf.Serialize(file, hashMap);

file.Close();

}

void LoadDictionary()

{//loads dictionary

if (File.Exists(savePath))

{

BinaryFormatter bf = new BinaryFormatter();

FileStream file = File.Open(savePath, FileMode.Open);

hashMap = (Dictionary<int, string>)bf.Deserialize(file);

file.Close();

}

}

private void Add(string value)

{

int key = GetHashCode(value); //gets hash code for position in dictionary

while (hashMap.ContainsKey(key))

{

key++; // Increment the key by 1 if collision occurs.

}

if (!hashMap.ContainsKey(key))

{//if there is known collision value is mapped to the key generated by hash

hashMap[key] = value;

}

}

public int getKey(string value)

{//used for finding a key given a value by doing the hash

try

{

int key = GetHashCode(value);

while (hashMap[key] != value && key <= 99)

{

key++;

}

if (key >= 99)//using mod 100 as the hashing algorithm, it means that the key cannot be greater than 99

{

Debug.Log("key not found");

return -1;//if key not found -1 is returned

}

return key;

}

catch

{

return(-1); //returns the null value -1 if the value is not found

}

}

private int GetHashCode(string value)

{

int hashCode = 0;

foreach (char c in value)

{//adds the ascii value of each letter in the string, adds it together and performs mod 100 to get the hash code

hashCode += (int)c;

}

hashCode = hashCode % 100;

return hashCode;

}

public Dictionary<int, string> GetHashMap()

{

return hashMap;//get hashmap function returns the hashmap dictionary.

}

}

## A screenshot of a web page Description automatically generatedHealth Bar

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class healthBar : MonoBehaviour

{

public Slider slider;

public Image fill;

public void setMaxHealth(int health)

{

slider.maxValue = health;

slider.value = health;

}

public void setHealth(int health)

{

slider.value = health;

}

}

## Item Database Object

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

[CreateAssetMenu(fileName = "New Database Item", menuName = "Inventory System/Items/Database")]

public class ItemDatabaseObject : ScriptableObject, ISerializationCallbackReceiver

{

public ItemObject[] ItemObjects;

public Dictionary<int, ItemObject> GetItem = new Dictionary<int, ItemObject>(); //gives and ID to every item using a dictionary

public void OnAfterDeserialize()

{// calls after the scritable object has been deserialized

for (int i = 0; i < ItemObjects.Length; i++)

{

//iterates through item objects and gives them an ID

//adds item object to the get item dictionary using the index as the key

ItemObjects[i].data.Id = i;

GetItem.Add(i, ItemObjects[i]);

}

}

public void OnBeforeSerialize()

{

//clears the dictionary before serialization

//this makes sure that data added during the onafterdesrialize method

//this prevents issues with duplicate keys

GetItem = new Dictionary<int, ItemObject>();

}

}

## Item Object

using JetBrains.Annotations;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public enum ItemType

{

Food,

Helmet,

Weapon,

Shield,

Boots,

ChestPlate,

Mobility,

Default

}

public abstract class ItemObject : ScriptableObject

{

public Sprite uiDisplay;

public bool stackable;

public ItemType type;

[TextArea(15,20)]

public string description;

public Item data = new Item();

}

[System.Serializable]

public class Item

{

public string Name;

public int Id = -1;

public Item()

{

Name = "";

Id = -1; //-1 Id for a null object

}

public Item(ItemObject item)

{

//every item will have a name and ID

Name = item.name;

Id = item.data.Id;

}

}

## Item Spawn Button

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ItemSpawnButton : MonoBehaviour

{// this script attaches to a game object to reveal another game object e.g. Button for parkour.

public GameObject grapple;

private void Start()

{

grapple.SetActive(false);

}

private void OnTriggerEnter(Collider collision)

{

if (collision.CompareTag("Player2"))

{

grapple.SetActive(true);

}

}

}

## A screenshot of a computer Description automatically generatedInventory Object

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using Unity.VisualScripting;

using UnityEngine;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using UnityEditor;

using System.Runtime.Serialization;

using JetBrains.Annotations;

using static UnityEditor.Progress;

using System.Runtime.CompilerServices;

using System.Linq;

[CreateAssetMenu(fileName = "New Inventory", menuName = "Inventory System/Inventory")]

public class InventoryObject : ScriptableObject

{

public string savePath;

public ItemDatabaseObject database;

public Inventory Container;

public InventorySlot[] GetSlots { get { return Container.Slots; } }

//This is a property that returns the array of inventory slots from the container called GetSlots

public bool AddItem(Item \_item, int \_amount)

{//adds item to inventory by amount specified

if(EmptySlotCount <= 0)

{

return false;

}

InventorySlot slot = FindItemOnInventory(\_item);

if (!database.GetItem[\_item.Id].stackable || slot == null)

{

SetEmptySlot(\_item, \_amount);

return true;

}

slot.AddAmount(\_amount);

return true;

}

public int EmptySlotCount

{ // counts how many empty slots there are by iteration

get

{

int counter = 0;

for(int i = 0; i< GetSlots.Length; i++)

{

if (GetSlots[i].item.Id <= -1)

{

counter++;

}

}

return counter;

}

}

public InventorySlot FindItemOnInventory(Item \_item)

{

//finds item in inventory by iteration

for(int i = 0;i< GetSlots.Length; i++)

{

if (GetSlots[i].item.Id == \_item.Id)

{

return GetSlots[i];

}

}

return null;

}

public InventorySlot SetEmptySlot(Item \_item, int \_amount)

{

//sets all slots to have id -1 by iteration

for (int i = 0; i < GetSlots.Length; i++)

{

if (GetSlots[i].item.Id <= -1)

{

GetSlots[i].UpdateSlot(\_item, \_amount);

return GetSlots[i];

}

}

return null;

}

public void SwapItems(InventorySlot item1, InventorySlot item2)

{//swapping two items

if(item2.CanPlaceInSlot(item1.ItemObject)&& item1.CanPlaceInSlot(item2.ItemObject))

{

InventorySlot temp = new InventorySlot(item2.item, item2.amount);

item2.UpdateSlot(item1.item, item1.amount);

item1.UpdateSlot(temp.item, temp.amount);

}

}

public int AmountOfItem(Item \_item)

{//finds amount of item using the information from the inventory slot class that each item inherits

if (Container != null)

{

InventorySlot slot = FindItemOnInventory(\_item);

if (slot != null)

{

return slot.amount;

}

}

return 0;

}

[ContextMenu("Save")]

public void Save()

{

IFormatter formatter = new BinaryFormatter(); //binary formatter so data is unreadable for humans

Stream stream = new FileStream(string.Concat(Application.persistentDataPath, savePath), FileMode.Create, FileAccess.Write);

//file stream for writing into text files, persistant path saves it to a path which is consistent between devices for the future of the game

formatter.Serialize(stream, Container);

//serializes container and this gets written into the text file

stream.Close();

}

[ContextMenu("Load")]

public void Load()

{

if (File.Exists(string.Concat(Application.persistentDataPath, savePath))) //check if file exists

{

IFormatter formatter = new BinaryFormatter(); //new instance of binary formatter for deserialization

Stream stream = new FileStream(string.Concat(Application.persistentDataPath, savePath), FileMode.Open, FileAccess.Read);

Inventory newContainer = (Inventory)formatter.Deserialize(stream);

//deserialization so that it is readable by unity

int minItemCount = Mathf.Min(GetSlots.Length, GetSlots.Length);

//Finds the minimum number of slots to iterate through to avoid index out-of-range errors

for (int i = 0; i < GetSlots.Length; i++)

{

GetSlots[i].UpdateSlot(GetSlots[i].item, GetSlots[i].amount);

}

//Iterates through the slots in the loaded

//inventory and updates the corresponding slots in the current container.

stream.Close();

}

}

[ContextMenu("Clear")]

public void Clear()

{

Container.Clear();

}

}

public delegate void SlotUpdated(InventorySlot \_slot);

[System.Serializable]

public class InventorySlot //represents every slot in the inventory

{

public ItemType[] AllowedItems = new ItemType[0];//allowed items which are set in the editor

[System.NonSerialized]

public UserInterface parent;//parent used so that it doesnt save and makes it so that it is empty

//reference to the interface the slot belongs to (either equipment or the inventory in this case)

[System.NonSerialized]

public GameObject slotDisplay;

// visuals

[System.NonSerialized]

public SlotUpdated OnAfterUpdate;

//event that gets triggered after update

[System.NonSerialized]//non serialized so cannot be seen by unity

public SlotUpdated OnBeforeUpdate;

//event that gets triggered before update

public Item item;

public int amount;

public InventorySlot()

{

UpdateSlot(new Item(), 0);

//default slot (with item id -1 and amount 0)

}

public InventorySlot(Item \_item, int \_amount)

{

UpdateSlot(\_item, \_amount);

//updates slot with the item and amount

}

public void UpdateSlot(Item \_item, int \_amount)

{

//invoke schedules methods for later times

if (OnBeforeUpdate != null) // != null used so that it doesnt keep running the function

// if it has already done it

{

OnBeforeUpdate.Invoke(this);

}

//the "this" keyword uses the current instant of the Inventory slot

//which is usefull as we have many inventory slots

item = \_item;

amount = \_amount;

//updates item and amount

if (OnAfterUpdate != null)

{

OnAfterUpdate.Invoke(this);

}

}

public void RemoveItem()

{

UpdateSlot(new Item(), 0);

//reusing the update slot function with different parameters

//so you dont need to repeat code with the same logic

}

public void AddAmount(int value)

{

UpdateSlot(item, amount += value);

//reusing the update slot function with different parameters

//so you dont need to repeat code with the same logic

}

public bool CanPlaceInSlot(ItemObject \_itemObject)

{//returns bool of if you can place in slot or not

if (AllowedItems.Length <= 0 || \_itemObject == null || \_itemObject.data.Id < 0)

{//if ther is nothing in the slot then you can place in it

return true;

}

for (int i=0; i <AllowedItems.Length; i++)

{//checks if item type matches with the allowed items list by iteration

if (\_itemObject.type == AllowedItems[i])

{

return true;

}

}

return false;

}

public ItemObject ItemObject

{

get

{

if(item.Id >= 0)

{

return parent.inventory.database.GetItem[item.Id];

}

return null;

}

//Returns the current item in the slot.

// this is because the parent contains the inventory's database so retrieves the item from there

}

}

[System.Serializable]

public class Inventory

{

public void Clear()

{//iterates and removes items from all slots

for(int i=0; i<Slots.Length; i++)

{

Slots[i].RemoveItem();

}

}

public InventorySlot[] Slots = new InventorySlot[16]; //used a list with specified size of 16 for the inventory

}

## A screenshot of a menu Description automatically generatedMain Menu

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour

{

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex +1);

}

public void PlayGameAgain()

{

SceneManager.LoadScene(1);

}

public void QuitGame()

{

Application.Quit();

}

}

## A screenshot of a computer Description automatically generatedMaze Cell

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class MazeCell : MonoBehaviour

{

//serialized fields for all

public GameObject leftWall;

public GameObject rightWall;

public GameObject frontWall;

public GameObject backWall;

public GameObject unvistedBlock;

public bool IsVisted { get; private set; }

public void Visit()

{

IsVisted = true;

unvistedBlock.SetActive(false);

}

public void ClearLeftWall()

{

leftWall.SetActive(false);

}

public void ClearRightWall()

{

rightWall.SetActive(false);

}

public void ClearFrontWall()

{

frontWall.SetActive(false);

}

public void ClearBackWall()

{

backWall.SetActive(false);

}

}

## A screenshot of a computer Description automatically generatedMaze Generation

using System.Collections;

using System.Linq; //used to get random numbers

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

// change IEnumerator to void then remove yield return

public class MazeGeneration : MonoBehaviour

{

public MazeCell mazeCell;

public int mazewidth;

public int mazeDepth;

public PlayerInventory playerInventory;

public Item keys;

private MazeCell[,] mazeGrid;

IEnumerator Start()

{

keys.Id = 7;

mazeGrid = new MazeCell[mazewidth, mazeDepth];

for (int x = 0; x < mazewidth; x++)

{

for (int z = 0 ; z < mazeDepth; z++)

{

mazeGrid[x, z] = Instantiate(mazeCell, new Vector3(x, 0, z), Quaternion.identity, transform);

mazeGrid[x, z].transform.localPosition = new Vector3(x, 0, z);

//loops through every position in mazeGrid creates a cell and stores in the array

//transform makes it able to scale depending on the parent game object

// however this made all the cells overlap so I had to use localPosition so that the uses world positions instead of positions local to the parent, also I had to add the line after to change local position to the correct value

Instantiate uses world position I want this to be the localposition to the parent object

}

}

mazeGrid[0, 0].ClearLeftWall(); //makes an entrance

mazeGrid[mazewidth - 1, mazeDepth - 1].ClearRightWall();//makes and exit

yield return GenerateMaze(null, mazeGrid[0, 0]);// clears entrance cell

}

private IEnumerator GenerateMaze(MazeCell previousCell, MazeCell currentCell)

{

currentCell.Visit();

ClearWalls(previousCell, currentCell);

yield return new WaitForSeconds(0.05f);

MazeCell nextCell;

do // problem of not goint to every cell in maze solved

{

nextCell = GetNextUnvisitedCell(currentCell);

if (nextCell != null)

{

yield return GenerateMaze(currentCell, nextCell); //use yield return to call co-routine

}

} while (nextCell != null);

}

private IEnumerable<MazeCell> GetUnvisitedCells(MazeCell currentCell)

{

// IEnumerable is a collection of objects that can be enumerated (iterated) sequentially

// in this case the maze cells

// this function checks all maze cells around the maze cell passed through the function

int x = (int)currentCell.transform.localPosition.x;

int z = (int)currentCell.transform.localPosition.z;

if (x + 1 < mazewidth)

{

var cellToRight = mazeGrid[x + 1, z];

if (cellToRight.IsVisted == false)

{

yield return cellToRight;

}

}

if (x - 1 >= 0)

{

var cellToLeft = mazeGrid[x - 1, z];

if (cellToLeft.IsVisted == false)

{

yield return cellToLeft;

}

}

if (z + 1 < mazeDepth)

{

var cellToFront = mazeGrid[x, z + 1];

if (cellToFront.IsVisted == false)

{

yield return cellToFront;

}

}

if (z - 1>= 0)

{

var cellToBack = mazeGrid[x, z - 1];

if (cellToBack.IsVisted == false)

{

yield return cellToBack;

}

}

}

private MazeCell GetNextUnvisitedCell(MazeCell currentCell)

{

var unvistedCells = GetUnvisitedCells(currentCell);

return unvistedCells.OrderBy(\_ => Random.Range(1, 10)).FirstOrDefault();

}

private void ClearWalls(MazeCell previousCell, MazeCell currentCell)

{

if (previousCell == null)

{

return;

}

if (previousCell.transform.localPosition.x < currentCell.transform.localPosition.x)

{

previousCell.ClearRightWall();

currentCell.ClearLeftWall();

return;

}

if (previousCell.transform.localPosition.x > currentCell.transform.localPosition.x)

{

previousCell.ClearLeftWall();

currentCell.ClearRightWall();

return;

}

if (previousCell.transform.position.z < currentCell.transform.position.z)

{

previousCell.ClearFrontWall();

currentCell.ClearBackWall();

return;

}

if (previousCell.transform.localPosition.z > currentCell.transform.localPosition.z)

{

previousCell.ClearBackWall();

currentCell.ClearFrontWall();

return;

}

}

private void Update()

{

if (playerInventory.InEquipment(keys))

{

SceneManager.LoadScene("Level3");

}

}

}

## Next Level Button

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.SceneManagement;

public class NextLevelButton : MonoBehaviour

{

public InventoryObject inventory;

private void OnTriggerEnter(Collider collision)

{

if (collision.CompareTag("Player"))

{//when player stands on button the next scene is loaded in the index

inventory.Clear();

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

}

}

## A screenshot of a computer Description automatically generatedNumber Game Controller

The Number Game Controller script controls the logic for the number game in the first level. Combat Cam script controls the combat camera view of the player. Combat Cam Toggle Button script controls the toggling between third person and combat cameras. The switching script controls the switching of the two players in the game.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using TMPro;

public class NumberGameController : MonoBehaviour

{

public GameObject parkourButton;

public TextMeshPro displayText;

public TextMeshPro displayText2;

private int targetNumber;

public bool player1OnButton;

public bool player2OnButton;

public AudioClip ringSound;

private AudioSource audioSource;

public int player1Number;

public int player2Number;

public int attempts;

void Start()

{

attempts = 0;

SpawnTargetNumber();

audioSource = GetComponent<AudioSource>();

audioSource.clip = ringSound;

}

void Update()

{

// Check if both players are standing on buttons

if (player1OnButton && player2OnButton)

{

int sum = player1Number + player2Number;

// Check if the sum matches the target number

if (sum == targetNumber)

{

displayText.color = Color.green;

displayText2.color = Color.green;

audioSource.Play();

StartNewRound();

}

}

}

void StartNewRound()

{

attempts++;

if (attempts <= 5)

{

SpawnTargetNumber();

}

else

{

parkourButton.SetActive(true);

gameObject.SetActive(false);

}

}

void SpawnTargetNumber()

{

targetNumber = Random.Range(2, 19);

displayText.color = Color.white;

displayText2.color = Color.white;

displayText.text = targetNumber.ToString();

displayText2.text = targetNumber.ToString();

}

}

## A screenshot of a computer program Description automatically generatedNumber On Button

Number On Button script checks the number inputted by the player on the first level and display score script displays the score at the end of the game.

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class NumberOnButton : MonoBehaviour

{// checks if player is on the button for the first level of the game and the input from the player is passed to the NumberGameController script

public int Number;

public bool playerOnButton;

public NumberGameController gameController;

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Player2"))

{

gameController.player2OnButton = true;

gameController.player2Number = Number;

}

if (other.CompareTag("Player"))

{

gameController.player1OnButton = true;

gameController.player1Number = Number;

}

}

private void OnTriggerExit(Collider other)

{

if (other.CompareTag("Player2"))

{

gameController.player2OnButton = false;

gameController.player2Number = 0;

}

if (other.CompareTag("Player"))

{

gameController.player1OnButton = false;

gameController.player1Number = 0;

}

}

}

## Options Menu

A screenshot of a computer program

Description automatically generated

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Audio;

using UnityEngine.UI;

using TMPro;

public class OptionsMenu : MonoBehaviour

{

public AudioMixer mixer;

Resolution[] resolutions;//array of available resolutions

public TMP\_Dropdown resolutionDropdown; //makes it a variable you can put in to the inspector

private void Start()

{

resolutions = Screen.resolutions;

resolutionDropdown.ClearOptions(); //clears current options on dropdown.

List<string> options = new List<string>();//list is an array with a changeable size in this case it contains the options

int currentResolutionIndex = 0;

for (int i=0; i<resolutions.Length; i++)//loop through each element and makes a user friendly string to display on the drop down

{

string option = resolutions[i].width + " x " + resolutions[i].height;

options.Add(option); // adds to option list for every i

if (resolutions[i].width == Screen.currentResolution.width && resolutions[i].height == Screen.currentResolution.height)

{//checks if current resolution is equal to current resolution of display so it displays automatically on drop down

currentResolutionIndex = i;

}

}

resolutionDropdown.AddOptions(options); // adds options list with every available resolution and displays it

resolutionDropdown.value = currentResolutionIndex;

resolutionDropdown.RefreshShownValue();

}

public void SetResolution(int resolutionIndex)

{

Resolution resolution = resolutions[resolutionIndex];

Screen.SetResolution(resolution.width, resolution.height, Screen.fullScreen);

}

public void SetVolume(float volume)

{

mixer.SetFloat("Volume", volume);

}

public void SetQuality(int qualityIndex)

{

QualitySettings.SetQualityLevel(qualityIndex);

}

public void SetFullscreen(bool isFullscreen)

{

Screen.fullScreen = isFullscreen;

}

}

## Pause Menu

A screenshot of a computer program

Description automatically generated

using JetBrains.Annotations;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.SceneManagement; //handles namespace within the unity scene view

public class PauseMenu : MonoBehaviour

{

public static bool GameIsPaused = false;

public GameObject pauseMenuUI;

private void Start()

{

Resume();

}

private bool wasPKeyPressed = false; //variable to track whether P was pressed

void Update()

{

if (Input.GetKey(KeyCode.P) && !wasPKeyPressed) //only when P is pressed

{

if (GameIsPaused)

{

Resume();

}

else

{

Pause();

}

}

//Update the was pressed key for the next frame

wasPKeyPressed = Input.GetKey(KeyCode.P);

}

public void Resume()

{

pauseMenuUI.SetActive(false);

Time.timeScale = 1f;

GameIsPaused = false;

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = false;

}

void Pause()

{

pauseMenuUI.SetActive(true); //sets Boolean parameters of menu UI to be true

Time.timeScale = 0f; //Pauses the time of the game

GameIsPaused = true;

Cursor.lockState = CursorLockMode.None;

Cursor.visible = true;

}

public void QuitGame()

{

Application.Quit();

}

public void LoadMenu()

{

Time.timeScale = 1f;

SceneManager.LoadScene("Menu"); //changes scene to menu

}

}

## A screenshot of a computer Description automatically generatedPlayer HP

PlayerHP controls the death and HP of players.

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

public class playerHP : MonoBehaviour

{

public int health = 100;

public healthBar healthBar;

public PlayerInventory inventory;

public bool dead;

private void Start()

{

healthBar.setMaxHealth(health);

}

public void TakeDamage(int damage)

{

if (inventory.helmetHealth > 0 || inventory.chestHealth > 0 || inventory.bootsHealth > 0)

{//takes damage off armour items first.

if(inventory.helmetHealth > 0)

{

inventory.helmetHealth -= damage;

if(inventory.helmetHealth <= 0)

{

inventory.RemoveItem(inventory.helmet);

}

return;

}

if(inventory.chestHealth > 0)

{

inventory.chestHealth -= damage;

if (inventory.chestHealth <= 0)

{

inventory.RemoveItem(inventory.chest);

}

return;

}

if(inventory.bootsHealth > 0 )

{

inventory.bootsHealth -= damage;

if (inventory.bootsHealth <= 0)

{

inventory.RemoveItem(inventory.boots);

}

return;

}

}

else if (health > 0)

{

health -= damage;

healthBar.setHealth(health);

}

else

{

Destroy(gameObject);

dead = true;

}

}

}

## Player Inventory

A screenshot of a computer

Description automatically generated

using JetBrains.Annotations;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using UnityEditor.SearchService;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.SocialPlatforms.Impl;

using static UnityEditor.Progress;

public class PlayerInventory : MonoBehaviour

{

public MovementScript movementScript;

public InventoryObject inventory;

public InventoryObject equipment;

public bool inventoryOpen;

public playerHP player;

public Item redBall;

public Item helmet;

public Item chest;

public Item boots;

public bool helmetDetected;

public bool chestDetected;

public bool bootsDetected;

public int helmetHealth = 0;

public int chestHealth = 0;

public int bootsHealth = 0;

public Item keys;

public Item food;

public Item bones;

public Canvas inventoryDisplay;

public ProjectileAddon projectileAddon;

public ThrowingBehaviour script;

public void OnControllerColliderHit(ControllerColliderHit hit)

{

var item = hit.collider.GetComponent<GroundItem>();

if (item)

{

Item \_item = new Item(item.item);

if(inventory.AddItem(\_item, 1))

{

Destroy(hit.gameObject);

}

}

}

private void Start()

{

bones.Id = 0;

boots.Id = 1;

chest.Id = 2;

food.Id = 3;

helmet.Id = 5;

keys.Id = 6; //initialises item ids to be used in the script

redBall.Id = 7;

inventoryDisplay.enabled = false;

inventoryOpen = false;

if (SceneManager.GetActiveScene().name == "Level3")

{

inventory.AddItem(redBall, 20);

}

}

private void Update()

{

//following checks for armour on equipment and controls health for each armour piece

if (InEquipment(chest) && !chestDetected)

{

chestHealth += 30;

chestDetected = true;

}

else if (!InEquipment(chest) && chestDetected)

{

chestHealth -= 0;

chestDetected = false;

}

if (InEquipment(helmet) && !helmetDetected)

{

helmetHealth += 30;

helmetDetected = true;

}

else if (!InEquipment(helmet) && helmetDetected)

{

helmetHealth = 0;

helmetDetected = false;

}

if (InEquipment(boots) && !bootsDetected)

{

bootsHealth += 30;

bootsDetected = true;

}

else if (!InEquipment(boots) && bootsDetected)

{

bootsHealth = 0;

bootsDetected = false;

}

if (inventory.AmountOfItem(keys) == 5 && SceneManager.GetActiveScene().name == "Level3")

{//checks number of keys in inventory and next level is loaded

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

if (inventory.AmountOfItem(keys) == 1 && SceneManager.GetActiveScene().name == "Level2")

{

inventory.Clear();

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

if (InEquipment(food))

{// controlls food regeneration

player.health += AmountInEquipment(food);

TakeAwayItem(food, AmountInEquipment(food));

RemoveItem(food);

}

if (Input.GetKey(KeyCode.K))

{//saving

inventory.Save();

equipment.Save();

}

if (Input.GetKeyDown(KeyCode.L))

{//loading

inventory.Load();

equipment.Load();

}

if (Input.GetKeyDown(KeyCode.I))

{//inventory opening logic.

if(inventoryOpen == true)

{

Time.timeScale = 1f;

}

else

{

Time.timeScale = 0f;

}

Cursor.lockState = (Cursor.lockState == CursorLockMode.None) ? CursorLockMode.Locked : CursorLockMode.None;

//toggles between cursorlockmode.none and cursorlockemode.locked

Cursor.visible = (Cursor.lockState == CursorLockMode.None);

//if cursor mode is in none then it makes the cursor visible

inventoryOpen = !inventoryOpen;

if (inventoryDisplay != null)

{

inventoryDisplay.enabled = !inventoryDisplay.enabled;

}

movementScript.enabled = !movementScript.enabled;

}

}

public bool InEquipment(Item Finditem)

{

// Loop through the 5 slots in the inventory

for (int i = 0; i < Mathf.Min(5, equipment.GetSlots.Length); i++)

{

InventorySlot currentSlot = equipment.GetSlots[i];

int itemId = currentSlot.item.Id;

if (itemId == Finditem.Id)

{

return true;

}

}

return false;

}

public int AmountInEquipment(Item item)

{//function which checks the number of the specified item in the inventory of the player

for (int i = 0; i < equipment.GetSlots.Length; i++)

{

InventorySlot currentSlot = equipment.GetSlots[i];

int itemId = currentSlot.item.Id;

if (itemId == item.Id)

{

return currentSlot.amount;

}

}

return 0;

}

public void TakeAwayItem(Item item, int amount) //takes away items

{

equipment.AddItem(item, -amount);

}

public void RemoveItem(Item item) //removes item GUI and logic from inventory container

{

InventorySlot slot = equipment.FindItemOnInventory(item);

slot.RemoveItem();

}

private void OnApplicationQuit()

{//clears inventory so players don’t load up with inventory from the last instance of the game

inventory.Container.Clear();

equipment.Container.Clear();

}

}

## A screenshot of a computer program Description automatically generatedPlayFab Manager

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using PlayFab;

using PlayFab.ClientModels;

using UnityEditor.PackageManager;

using UnityEngine.UI;

using TMPro;

using System.Security.Cryptography;

using System.Text;

using UnityEngine.SceneManagement;

using Unity.VisualScripting;

public class PlayfabManager : MonoBehaviour

{

public static PlayfabManager instance;

public TextMeshProUGUI messageText;

public TMP\_InputField email;

public TMP\_InputField password;

void Awake()

{

instance = this;

DontDestroyOnLoad(gameObject); //so the user stays logged in the entire game

}

public void RegisterButton()

{//when register button is pressed this function runs

if (password.text.Length < 6)

{

messageText.SetText("password too short (min 6 characters)");

return;

}

var request = new RegisterPlayFabUserRequest

{//request to register player with

Email = email.text,

Password = HashingAlg(password.text), //calculates a hash and stores it in the server

RequireBothUsernameAndEmail = false

};

PlayFabClientAPI.RegisterPlayFabUser(request, OnRegisterSuccess, OnError);

}//sends a request to the Playfabclient API to register the player

void OnRegisterSuccess(RegisterPlayFabUserResult result)

{ //when register is succesful the menu scene is loaded

messageText.SetText("Registered and Logged in");

SceneManager.LoadScene(0);//loads up starting menu

}

public string HashingAlg(string input)

{

int hash = 485675842;//the bigger the number the harder to crack the hash

foreach(char c in input)

{

hash = ((hash << 5) + hash) + c; //for every character in the input, the hash's binary value is moved left 5 spaces meaning it gets multiplied by 32 and adds the previous hash and the ascii value of the character in input

}

return hash.ToString("X");//converts to hexadecimal

}

public void Login()

{

var request = new LoginWithEmailAddressRequest

{

Email = email.text,

Password = HashingAlg(password.text),

//checks with match of hashed password in playfab server

};

PlayFabClientAPI.LoginWithEmailAddress(request, OnSuccess, OnError);

}

void OnSuccess(LoginResult result)

{

messageText.SetText("Logged in");

SceneManager.LoadScene(0); //loads menu scene when succesfully logged in

}

void OnError(PlayFabError error)

{

messageText.SetText(error.ErrorMessage);

//error message displayed to user

}

public void SendLeaderboard(int score)

{//every time a score is added the score is sent to the server to make a leaderboard

var request = new UpdatePlayerStatisticsRequest

{

Statistics = new List<StatisticUpdate>

{

new StatisticUpdate

{

StatisticName = "Score",

Value = score

}

}

};

PlayFabClientAPI.UpdatePlayerStatistics(request, OnLeaderboardUpdate, OnError);

}

void OnLeaderboardUpdate(UpdatePlayerStatisticsResult result)

{

Debug.Log("Succesfull leaderboard sent");

}

public void ResetPasswordButton()

{//when a user wants to reset password playfab will deal with sending them a password reset email

var request = new SendAccountRecoveryEmailRequest

{

Email = email.text,

TitleId = "273C8"

};

PlayFabClientAPI.SendAccountRecoveryEmail(request, OnPasswordReset, OnError);

}

private void OnPasswordReset(SendAccountRecoveryEmailResult result)

{

messageText.SetText("Password reset email sent");

}

}

## Projectile Addon

A screenshot of a computer program

Description automatically generated

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ProjectileAddon : MonoBehaviour

{//logic for the ball that gets shot from player

public int damage = 5;

private Rigidbody rb;

private float instantiateTime;

private void Start()

{

rb = GetComponent<Rigidbody>();

instantiateTime = Time.time;

}

private void OnCollisionEnter(Collision collision)

{//when hitting something the game object is destroyed

Destroy(gameObject);

}

private void OnTriggerEnter(Collider other)

{

Enemy1 enemy1 = other.GetComponent<Enemy1>();

if (enemy1 != null)

{//takes damage if the ball hits an enemy

enemy1.TakeDamage(damage);

}

// make sure projectile sticks to surface

rb.isKinematic = true;

// make sure projectile moves with target

transform.SetParent(other.transform);

}

private void Update()

{

if (Time.time - instantiateTime > 5f)

{//destroys game object so after 5 seconds so that the ball doesn’t keep flying if it is shot into the distance and doesn’t hit anything this means there is less unnecessary processing power.

Destroy(gameObject);

}

}

}

## Projectile Addon E1

A screenshot of a computer program

Description automatically generated

using System;

using System.Collections;

using System.Collections.Generic;

using Unity.VisualScripting;

using UnityEngine;

public class ProjectileAddonE1 : MonoBehaviour

{//logic for the projectile that gets thrown by the enemy

public int damage = 5;

private float instantiateTime;

private void Start()

{

instantiateTime = Time.time;

}

private void OnCollisionEnter(Collision collision)

{

// Check if the collision is with an object named "player1"

if (collision.gameObject.name == "player1")

{

playerHP playerHP = collision.gameObject.GetComponent<playerHP>();

playerHP.TakeDamage(damage);

}

Destroy(gameObject);

}

private void Update()

{

if(Time.time - instantiateTime> 5f)

{

Destroy(gameObject);

}

}

}

## Spawn Items

Spawn Items script controls spawning items in the third level of the game. Player Inventory holds logic of the players inventory and equipment screens.

A screenshot of a computer

Description automatically generated

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class environmentScript : MonoBehaviour

{

public ItemDatabaseObject database;

public GroundItem itemPrefab;

public int itemsToBeSpawned;

public GameObject enemyPrefab;

public GameObject bossPrefab;

private int enemycount;

private int bosscount;

public playerHP p1;

public playerHP p2;

// Start is called before the first frame update

void Start()

{

for (int i = 0; i < itemsToBeSpawned ; i++)

{

var RandomValueZ = UnityEngine.Random.Range(81, 213);

var RandomValueX = UnityEngine.Random.Range(111, 282);

var RandomId = UnityEngine.Random.Range(0, 8);

ItemObject item =database.GetItem[RandomId];

itemPrefab.item = item;

Instantiate(itemPrefab, new Vector3(RandomValueX, 0, RandomValueZ), Quaternion.identity);

}

StartCoroutine(SpawnEnemy());

StartCoroutine(SpawnBoss());

}

IEnumerator SpawnEnemy()

{

while (enemycount < 25)

{

Instantiate(enemyPrefab, new Vector3(111f, 1f, 81f), Quaternion.identity);

Debug.Log("enemy spawned");

bosscount++;

yield return new WaitForSeconds(50); //every 50 seconds enemy is spawned

}

}

IEnumerator SpawnBoss()

{

while (bosscount < 2)

{

Instantiate(bossPrefab, new Vector3(200f, 0f, 95f), Quaternion.identity);

bosscount++;

yield return new WaitForSeconds(200); //enemy boss spawns every 200 seconds

}

}

private void Update()

{

if(p1.dead && p2.dead)

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);

}

}

}

## A screenshot of a computer program Description automatically generatedStatic Interface

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.EventSystems;

public class StaticInterface : UserInterface //this is attached to the equipment inventory so needs the same logic from user interface and needs to inherit some

methods

{

public GameObject[] slots;

public override void CreateSlots()

{

slotsOnInterface = new Dictionary<GameObject, InventorySlot>(); // contains information on which game object is in which equipment slot link database and inventory items

for (int i = 0; i < inventory.GetSlots.Length; i++)

{

var obj = slots[i];

AddEvent(obj, EventTriggerType.PointerEnter, delegate { OnEnter(obj); }); //called when entering slot

AddEvent(obj, EventTriggerType.PointerExit, delegate { OnExit(obj); }); //called when exiting slot

AddEvent(obj, EventTriggerType.BeginDrag, delegate { OnDragStart(obj); }); //called when starting

//to item from slot

AddEvent(obj, EventTriggerType.EndDrag, delegate { OnDragEnd(obj); }); //called at the end of the drag

AddEvent(obj, EventTriggerType.Drag, delegate { OnDrag(obj); }); //called when dragging object.

inventory.GetSlots[i].slotDisplay = obj; //updates display for inventory slot

slotsOnInterface.Add(obj, inventory.GetSlots[i]); // adds the item and its inventory slot to the dictionary

}

}

}

## Switching

Switching script used for toggling between the view of the two characters in the game when Tab is pressed.

A screenshot of a computer

Description automatically generated

using JetBrains.Annotations;

using System.Collections;

using System.Collections.Generic;

using System.Xml.Linq;

using Unity.VisualScripting;

using UnityEngine;

using UnityEngine.TextCore.Text;

public class Switching : MonoBehaviour

{

public Canvas player2inventory;

public GameObject character1;

public GameObject character2;

public bool p1;

public bool p2;

public Canvas healthbar1;

public Canvas healthbar2;

public Cinemachine.CinemachineFreeLook activeCameraFreeLook;

public Cinemachine.CinemachineVirtualCamera activeCameraCombat;

void Start()

{

healthbar1.enabled = true;

healthbar2.enabled = false;

p1 =true;

p2 =false;

Toggle();

Toggle();

player2inventory.enabled = false;

//fixes bug with the inventory being enabled at the start of the scene

}

void Update()

{

if (Input.GetKeyDown(KeyCode.Tab))

{

Toggle();

}

}

void Toggle()

{

p1 = !p1;

p2 = !p2;

if(p1 == false)

{

healthbar1.enabled = false;

healthbar2.enabled = true;

Components(character1, false);

Cameras(character2);

Components(character2, true);

}

else

{

healthbar1.enabled = true;

healthbar2.enabled = false;

Components(character2, false);

Cameras(character1);

Components(character1, true);

}

}

private void Components(GameObject character, bool state)

{

character.GetComponent<Animator>().SetBool("isGrounded", true);//disable all animations except idle

character.GetComponent<Animator>().SetBool("isJumping", false);

character.GetComponent<Animator>().SetBool("isFalling", false);

character.GetComponent<Animator>().SetBool("isMoving", false);

character.GetComponent<MovementScript>().enabled = state;

//character.GetComponent<CharacterController>().enabled = state;

character.GetComponent<ThrowingBehaviour>().enabled = state;

character.GetComponent<Grappling>().enabled = state;

character.GetComponent<CombatCamera>().enabled = state;

character.GetComponent<CombatCamToggleButton>().enabled = state;

character.GetComponent<PlayerInventory>().enabled = state;

}

private void Cameras(GameObject character)

{//enabling the cameras and make them follow the player

activeCameraCombat.Follow = character.transform.Find("CombatLookAt");

activeCameraCombat.LookAt = character.transform.Find("CombatLookAt");

activeCameraFreeLook.Follow = character.transform;

activeCameraFreeLook.LookAt = character.transform;

activeCameraCombat.enabled = true;

activeCameraFreeLook.enabled = false;

}

}

## Throwing Behaviour

A screenshot of a computer program

Description automatically generated

using System;

using System.Collections;

using System.Collections.Generic;

using System.Data;

using Unity.VisualScripting;

using UnityEngine;

public class ThrowingBehaviour : MonoBehaviour

{

[Header("References")]

public Transform cam;

public Transform attackPoint;

public GameObject objectToThrow;

public PlayerInventory playerInventory;

public Item redBalls;

[Header("Settings")]

public int totalThrows;

public float throwCooldown;

[Header("Throwing")]

public KeyCode throwKey = KeyCode.Mouse0;

public float throwForce;

public float throwUpwardForce;

bool readyToThrow;

private void Start()

{

readyToThrow = true;

}

private void Update()

{

try//code tries checking if there is any redballs in equipment

{

totalThrows = playerInventory.equipment.AmountOfItem(redBalls);

}

catch (NullReferenceException)//if there is an error then the code does not throw any balls

{

return;

}

if (Input.GetKeyDown(throwKey) && readyToThrow && totalThrows > 0 && playerInventory.inventoryOpen == false)

{//conditions for throwing

Throw();

playerInventory.TakeAwayItem(redBalls, 1);

}

else if (totalThrows == 0)

{

try

{

playerInventory.RemoveItem(redBalls);

}

catch (NullReferenceException)

{

return;

}

}

}

private void Throw()

{

readyToThrow = false;

// instantiate onject to throw

GameObject projectile = Instantiate(objectToThrow, attackPoint.position, cam.rotation);

// get rigidbody component

Rigidbody projectileRb = projectile.GetComponent<Rigidbody>();

// calculate direction using ray cast

Vector3 forceDirection = cam.forward;

RaycastHit hit;

if (Physics.Raycast(cam.position, cam.forward, out hit, 500f))

{

forceDirection = (hit.point - attackPoint.position).normalized;

}

//add force

Vector3 forceToAdd = forceDirection \* throwForce + transform.up \* throwUpwardForce;

projectileRb.AddForce(forceToAdd, ForceMode.Impulse);

totalThrows--;

//implement throw cooldown

Invoke(nameof(ResetThrow), throwCooldown);

}

private void ResetThrow()

{

readyToThrow = true;

}

}

## A screenshot of a computer program Description automatically generatedUser Interface

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

using UnityEngine.EventSystems;

using UnityEngine.Events;

using UnityEditor;

using System;

public abstract class UserInterface : MonoBehaviour

{

public InventoryObject inventory;

public Vector2 mouseinputPosition;

public Dictionary<GameObject, InventorySlot> slotsOnInterface = new Dictionary<GameObject, InventorySlot>();

void Start()

{

//create slots for items and sets up pointer enter and exit events for when the mouse goes in and out

//of a slot

for (int i = 0; i < inventory.GetSlots.Length; i++)

{

inventory.GetSlots[i].parent = this;

inventory.GetSlots[i].OnAfterUpdate += OnSlotUpdate;

}

CreateSlots();

AddEvent(gameObject, EventTriggerType.PointerEnter, delegate { OnEnterInterface(gameObject); });

AddEvent(gameObject, EventTriggerType.PointerExit, delegate { OnExitInterface(gameObject); });

}

public abstract void CreateSlots();//abstract void used here so that it can be overridden in the Static interface script. Serves as a blueprint.

private void OnSlotUpdate(InventorySlot \_slot) //updates visuals when slot gets updated.

{

if (\_slot.item.Id >= 0)

{

//updates to the sprite only this makes it less memory heavy

//graphical parts of item are in database and the system part is in the Item class

\_slot.slotDisplay.transform.GetChild(0).GetComponentInChildren<Image>().sprite = \_slot.ItemObject.uiDisplay;

\_slot.slotDisplay.transform.GetChild(0).GetComponentInChildren<Image>().color = new Color(1, 1, 1, 1);

//pure white 100% alpha for an empty slot (we can see the item)

\_slot.slotDisplay.GetComponentInChildren<TextMeshProUGUI>().text = \_slot.amount == 1 ? "" :

\_slot.amount.ToString("n0");

//ternary operator used here if slots value amount == 1 then don’t display any text if not 1

//display the number of items in that slot

}

else

{

\_slot.slotDisplay.transform.GetChild(0).GetComponentInChildren<Image>().sprite = null;

\_slot.slotDisplay.transform.GetChild(0).GetComponentInChildren<Image>().color = new Color(1, 1, 1, 0);

//alpha 0 so we cannot see the item

\_slot.slotDisplay.GetComponentInChildren<TextMeshProUGUI>().text = "";

}

}

//adds an event trigger to a gameobject, allowing the registration for different events

// protected means it is private but only available to the class its in and any class

// that inherits from it

protected void AddEvent(GameObject obj, EventTriggerType type, UnityAction<BaseEventData> action)

{

EventTrigger trigger = obj.GetComponent<EventTrigger>(); //get even trigger from game object

var eventTrigger = new EventTrigger.Entry(); //create new event trigger for the event passed through

eventTrigger.eventID = type;

eventTrigger.callback.AddListener(action); // this happens when the specified even happens this.

// means you don’t have to loop through every inventory slot

// to detect an event

trigger.triggers.Add(eventTrigger);

}

//the following are the methods for the mouse on each slot and Interface

public void OnEnter(GameObject obj)

{

MouseData.slotHoveredOver = obj;

}

public void OnExit(GameObject obj)

{

MouseData.slotHoveredOver = null;

}

public void OnExitInterface(GameObject obj)

{

MouseData.interfaceMouseIsOver = null;

}

public void OnEnterInterface(GameObject obj)

{

MouseData.interfaceMouseIsOver = obj.GetComponent<UserInterface>();

}

public void OnDragStart(GameObject obj)

{

MouseData.tempItemBeingDragged = CreateTempItem(obj); // at the start the object is duplicated

}

public GameObject CreateTempItem(GameObject obj) // creates temp item for the visual effect that follows the mouse

{

GameObject tempItem = null;

if (slotsOnInterface[obj].item.Id >= 0)

{

tempItem = new GameObject();

var rt = tempItem.AddComponent<RectTransform>();

rt.sizeDelta = new Vector2(50, 50);

tempItem.transform.SetParent(transform.parent);

var img = tempItem.AddComponent<Image>();

img.sprite = slotsOnInterface[obj].ItemObject.uiDisplay;

img.raycastTarget = false;

}

return tempItem;

}

public void OnDragEnd(GameObject obj)// handles removing of item and swapping (if there is any need) between the two slots.

{

Destroy(MouseData.tempItemBeingDragged);

if(MouseData.interfaceMouseIsOver == null)

{

slotsOnInterface[obj].RemoveItem();

return;

}

if (MouseData.slotHoveredOver)

{

InventorySlot mouseHoverSlotData = MouseData.interfaceMouseIsOver.slotsOnInterface

[MouseData.slotHoveredOver];

inventory.SwapItems(slotsOnInterface[obj], mouseHoverSlotData);

}

}

public void OnDrag(GameObject obj) //updates the position during dragging to follow the mouse.

{

if (MouseData.tempItemBeingDragged != null)

{

MouseData.tempItemBeingDragged.GetComponent<RectTransform>().position = Input.mousePosition;

}

}

}

public static class MouseData //class containing the mouse data.

{

public static UserInterface interfaceMouseIsOver;

public static GameObject tempItemBeingDragged;

public static GameObject slotHoveredOver;

}

public static class ExtensionMethods //extension method used so you can add methods on to the dictionary slotsOnInterface

{

public static void UpdateSlotDisplay(this Dictionary<GameObject, InventorySlot> \_slotsOnInterface)

{

foreach (KeyValuePair<GameObject, InventorySlot> \_slot in \_slotsOnInterface) //itterates through dictionary where gameobject is a key

it returns an inventory slot

{

if (\_slot.Value.item.Id >= 0)// if an item is the slot

{

//updates to the sprite only this makes it less memory heavy

//graphical parts of item are in database and the system part is in the Item class

\_slot.Key.transform.GetChild(0).GetComponentInChildren<Image>().sprite = \_slot.Value.ItemObject.uiDisplay;

\_slot.Key.transform.GetChild(0).GetComponentInChildren<Image>().color = new Color(1, 1, 1, 1);

//pure white 100% alpha for an empty slot (we can see the item)

\_slot.Key.GetComponentInChildren<TextMeshProUGUI>().text = \_slot.Value.amount == 1 ? "" :

\_slot.Value.amount.ToString("n0"); //displaying the value as a string

//ternary operator used here if slots value amount == 1 then don’t display any text if not 1 the amount of items in that slot

}

else

{

\_slot.Key.transform.GetChild(0).GetComponentInChildren<Image>().sprite = null;

\_slot.Key.transform.GetChild(0).GetComponentInChildren<Image>().color = new Color(1, 1, 1, 0);

//alpha 0 so we cannot see the item

\_slot.Key.GetComponentInChildren<TextMeshProUGUI>().text = "";

//and no text displayed

}

}

}

}

# Evaluation

## Final Interview

Below are a couple of questions I asked to my client (Alan Thomas) regarding the requirements of the end user.

1. **Are you satisfied with the result of the final game?**

Yes, I like that the game was easy to follow and simplistic. However, I didn’t find it too easy that it was boring. Some levels were difficult therefore making it fun to solve. For example, it took me a while to figure out how the first level worked but it was fun trying to figure it out by myself.

The grappling ability was quite fun to use, and the inventory system gave the game more freedom as I had control on what I can choose to use and when to use it. The movement of the automated enemies made the game more realistic than if the enemies were standing still. This added extra complexity to the game which kept me engaged.

1. **What improvements/changes would you have liked to see?**

If there was an option to play online it would have made it more enjoyable as I like to play with friends as I see gaming as a social activity. I would have liked 1 or 2 more problem solving levels as I found those most enjoyable. Also, the graphics could have been more complex (for example floor and wall textures and skins for enemies).

Although at the initial interview I preferred a linear progression, for the future I would like to see an open world level (maybe for the last level) as this adds more freedom for the player.

I would say that I have achieved the main aspects of the success criteria, and my client is happy with the experience the game has provided him. The use of the Cinemachine Camera’s paired with a character controller and efficient and well-designed code has produced an easy to use and smooth feeling movement experience (like an actual game).

The main menu has a variety of features like options to change the resolution of the game and the graphics quality. There are also information pages if the player gets stuck whilst playing the game.

The animations have further made the game look and feel better. This is important for my game as most people prioritised a well-rounded movement experience over other aspects of the game such as level quality and graphics. My client agreed with this.

The use of unique abilities like grappling are not traditional and are quite niche in a plain third person platformer game. These abilities (although are simple) are not too simple for a new gamer to understand but neither too complex to confuse the player. I implemented SUVAT equations for accurate physics for the grappling ability.

I have also created a procedurally generated level (the maze level) this has made the game interactive and replay able as every maze is different. The other levels have the goal of players working together towards a common goal. This can make the game more enjoyable to a new gamer who might be playing games as a social event rather than for improving their skills.

I have created a login and register system. The is a client server system where login details are stored on a server (PlayFab) and when the user gets a score at the end of the game it is added to a leaderboard. However, I wish to implement more customisation with each person who logs into the game e.g. changing the characters skin or a level designer where the player can design and publish their own levels.

Although I didn’t manage to make the game a multiplayer game, I kept the idea of working together as a team with a switching feature which switches between two players. My client could find this just as enjoyable as playing with a friend online as you can physically play with a friend every time you switch, or you can play by yourself. In my opinion, this has made the game more diverse in the way you can play it as it is not reliant on having another person to play with you. As a new gamer, playing with competitive friends may be intimidating so although it was not planned, this is a good feature.

To improve on, I could have made my scripts more organised in files for the future development of the project. Also, I didn’t get to implement the procedural dungeon generation of rooms that I wanted to use for the third level as the code did not work as planned. To make a complete game (which was one of my success criteria) I handcrafted the last level instead. The levels in the game look quite plain as I didn’t use many external Unity’s texture packs. Level design improvements can be made by using the current layout as a base for future development.

I wasn’t able to implement a HashMap within the Playfab servers for faster searching and more efficient sorting, but I have successfully coded the logic behind it. For the future I would like to add this so that it works with the PlayFab servers.

Interactive enemies have made the game fun, the use of Nav Mesh which implements pathfinding algorithms has made the game feel more realistic compared to some other games where NPC’s go in random directions and don’t react to the players position. Although the enemies are challenging to defeat, it is not too challenging and should be engaging enough for a new gamer to play against but not too hard that the player gives up.

Overall, the game design has been well thought out to fit a new gamer such as my client and get them inspired and excited about gaming whilst keeping a good balance between complexity and simplicity for a new gamer.