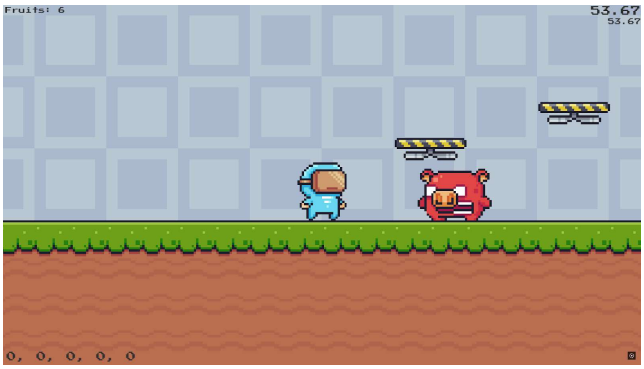
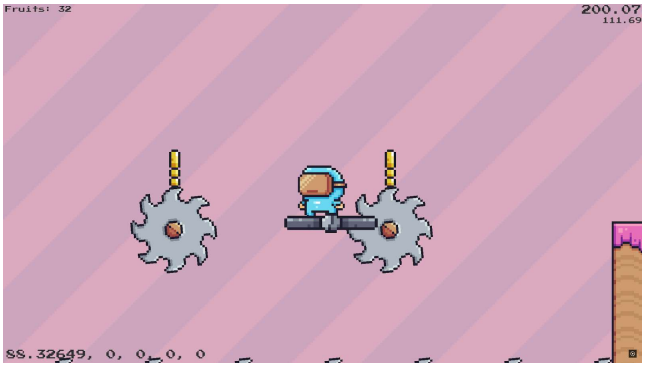
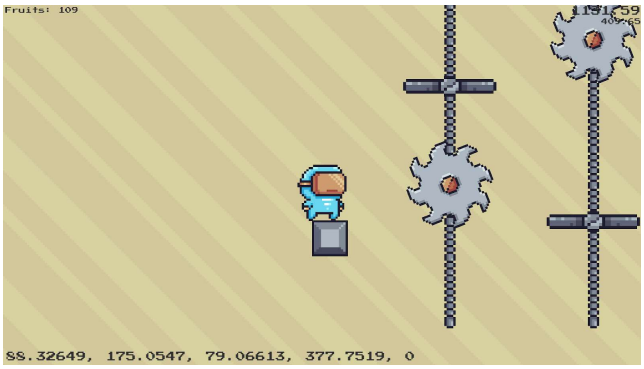
Super Pixel World

By James Worsley







Video: https://www.youtube.com/watch?v=eI9u33uxcCE

Analysis

Documented Design

Technical Solution

Testing

Evaluation

Analysis

Problem

In the current video game industry, there has been a lack of good 2D platformers that have been popular in the last ten years. I know my dad is looking for a 2D platformer to play but must resort to older ones because there have not been many since the 8 bit and 16 bit era. Therefore, my dad will be the client for my game.

Secondary research

The current market

In the top 100 games on metacritic[1] there is only one 2D platformer, but it was released in 2003. Also, the list does not contain games from the 16 bit era from before, which is when the majority of 2D platformers were released. This means that there have not been any top level 2D platformers in the last 20-30 years. This means that people who enjoy these games are often limited to older games, which can be hard to acquire and only have a limited quantity due to there only being a limited number of physical copies of the game. In addition, the majority of the best 2D platformers[2] tend to be older. Most are not from the last ten years and many are not from the last 20 years, because most consoles were limited to 2D in the 8 bit and 16 bit era. The most notable exception to this would be Celeste[3] (released in 2018) which itself is starting to become a bit old. Before Celeste was released the only big 2D platformers being released were Donkey Kong Country series, New Super Mario Bros series and Rayman games. However, these were released in the late 2000s and early 2010s. Whilst they were once new to me, these days many of them are reaching ten years old and have had no real successors other than Celeste. Despite Celeste being an indie game, it is still very big amongst 2D platforming enthusiasts and achieves very strong reviews that are better than those of games with a much larger budget[4].

Previous 2D platforming games I have played and researched:

- Super Mario World - Good control system that allows precise control of the player
- Celeste - Allows the player to dash in mid air, 16 bit art style despite 2018 release

---

[1] Metacritic top games https://www.metacritic.com/browse/game/?releaseYearMin=1958&releaseYearMax=2024&page=1 Accessed 13/05/24
[2] Top 2D platforming games (IGN) https://www.ign.com/playlist/argonmaster/lists/best-2d-platformers Accessed 13/05/24
[3] Celeste https://en.wikipedia.org/wiki/Celeste_(video_game) Accessed 13/05/24
[4] Celeste review https://www.metacritic.com/game/celeste/ Accessed 22/05/24

Evaluation of current games

Game	Features and analysis	Advantages	Disadvantages
Super Mario World[5] (most also applies to all 2D Mario games[6])	• Good control system that allows precise control of the player using the d-pad • Multiple worlds each made up of separate levels • Boss fights at the end of each world (and at mid-points of each world) • Underwater levels • Emulation of Super Mario World (and NES Mario Games) on the Nintendo Switch allows the player to recall the state of the game to various points the emulator chooses • End of game boss fights	• Precise control system • Improved on the original 3 2D mario games in both features and scale • Still played to this day despite 1990 release • Art style still used in more recent 2D platformers such as Celeste	• Arguably limited by hardware at the time • Newer 2D mario games haven't really made any innovations (New Super Mario Bros series)
Donkey Kong Country[7] (new and old games)	• Older games developed by Rare despite Nintendo owning	• 16-bit art style looks great and is unique • Good difficulty progression	• No 2D Donkey Kong platformers between 1996

[5] Super Mario World Nintendolife review https://www.nintendolife.com/reviews/snes/super_mario_world Accessed 3/6/24
[6] New Super Mario Bros Wii Metacritic review https://www.metacritic.com/game/new-super-mario-bros-wii/ Accessed 3/6/24
[7] Donkey Kong Country https://en.wikipedia.org/wiki/Donkey_Kong#Donkey_Kong_Country Accessed 05/02/2025

	the Donkey Kong franchise • SNES Donkey Kong Country games had impressive 16-bit art for the time and tried to have a 3D feel • Good difficulty progression • Good boss fights • More unique art and level design than Super Mario games	• Good boss fights • Unique level design	and 2010 non-inclusive • Not as popular as Mario games
Celeste[4]	• Allows the player to dash in mid air • 16 bit art style despite 2018 release (now with 16:9 aspect ratio) • Longer levels for each part of the mountain	• Was the biggest 2D platformer released in somewhat recent years, arguably reviving the genre • 16 bit art style looks great on modern TVs and monitors • Dash feature is something I have never seem implemented before • 92 critic score on Metacritic	• Arguably limited by smaller production team and budget
Rayman Origins[8]/Legends[9]	• Unlockable movement skills such as	• Successful competitor to New Super	• Hasn't had a successor in over ten years

[8] Rayman Origins Metacritic review https://www.metacritic.com/game/rayman-origins/ Accessed 3/6/24
[9] Rayman Legends Metacritic review https://www.metacritic.com/game/rayman-legends/ Accessed 3/6/24

	hovering, wall running and attack • Multiple worlds each made up of separate levels • Underwater levels • Music levels • Boss fights	Mario Bros series in the late 2000s and early 2010s • Added many new features that I haven't seen before • Lots of content in the game, fells very polished	
Fireboy and Watergirl[10]	• Playable on websites so don't need a specific console • Two player problem solving • Each character is killed if it touches the opposite "element" • Many sequels • Used to be very popular among my peers	• Playable on almost every computer (through a web browser) • Popular so many sequels • Unique two player problem solving	• Not available on console • Low production budget compared to other games analysed
Geometry Dash[11]	• Fast paced platforming based on best of music • Precision platforming because it has tight jumps • Rhythm based so has music • Community made levels	• Replayable with community made levels • Precision platforming and often hard difficulty makes level completion satisfying • Entertaining because of fast speed platforming and music	• Often only known as a mobile game, PC version not talked about as much • Only playable as intended with sound

[10] Fireboy and Watergirl series https://en.wikipedia.org/wiki/Fireboy_and_Watergirl Accessed 11/07/24
[11] Geometry Dash Metacritic review https://www.metacritic.com/game/geometry-dash/ Accessed 11/07/24

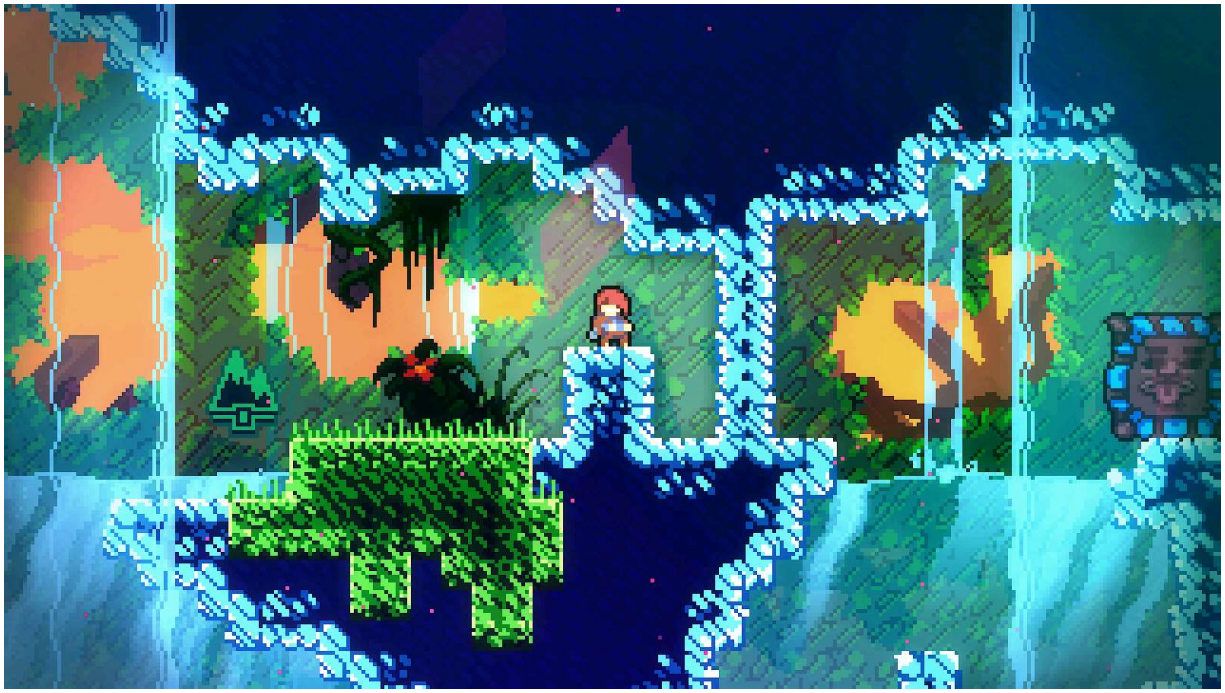
Super Mario World

New Super Mario Bros U

Rewind feature on Nintendo Switch (Super Mario Brothers 3)

Donkey Kong Country (original on SNES)

Donkey Kong Country Tropical Freeze (newer game on Wii U and Nintendo Switch)



Celeste

Rayman Origins

Rayman Legends

Geometry Dash

Game development lecture

Before starting development, I decided to attend a lecture at the University of Southampton about how to start creating your first game. Some important things I learned were:

- Choose a game engine that uses a simple language such as Unity and Godot.
- Make sure to get started sooner rather than later.
- Don’t be afraid to make mistakes because learning from them is important.
- Don't leave it too long before creating the first prototype of the game so you can keep making progress.
- See the game from the player’s point of view and not just from an algorithmic point of view.
- Finite state machines are very useful for representing a system (this example was for enemy behaviour).
- Start with a more simple game such as a 2D platformer instead of a complicated 3D game.

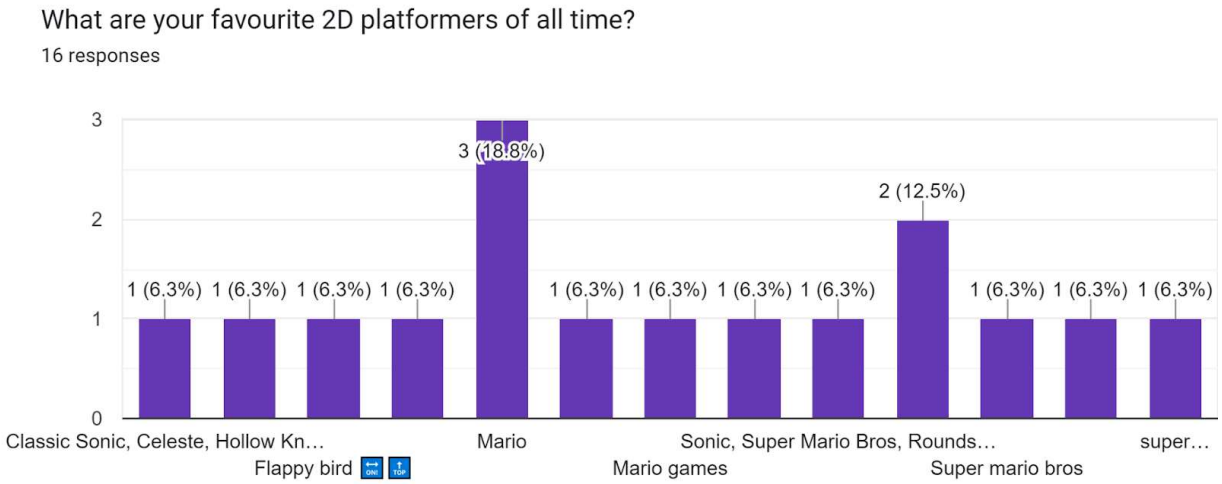
Game engine and programming language

To produce a 2D platformer the three main game engines available to the public (many large developers make their own) are Unreal Engine[12], Unity[13], and Godot[14]. While Unreal Engine has been used to create many amazing games such as[15] Fortnite and the Mass Effect series, it utilises C++ for scripts which is a lower-level high-level language. While this can be good for performance, this would be too time consuming to learn for this project. Godot is very beginner friendly, allowing the developer to add many game features in the game engine itself. Whilst this makes development easier, I would like to have more focus on the code itself to improve at object-oriented programming. Godot also uses its own “GDScript” which is similar to python. However, I would like to use a more widely used programming language so I can carry over this knowledge to other projects in the future. Lastly, Unity is a good balance between these two because many tedious tasks can be handled in the editor with a graphical interface, there is still lots of places for C# scripts to be utilised. C# is a good programming language to use because it is commonly used for object-oriented programming and is also very similar to Java so should help me with other projects too.

Primary research:

Class Questionnaire:

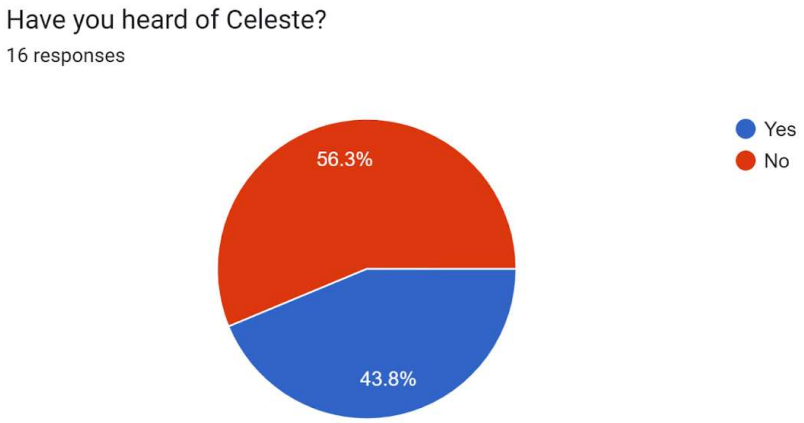
Q1



[12] Unreal Engine https://www.unrealengine.com/en-US Accessed 11/07/24
[13] Unity https://unity.com/ Accessed 11/07/24
[14] Godot https://godotengine.org/ Accessed 11/07/24
[15] Unreal Engine games https://en.wikipedia.org/wiki/Category:Unreal_Engine_games Accessed 11/07/24

The majority of responses are Mario games, which makes sense being that it is one of the most renowned franchises for 2D platformers. In particular, Super Mario World came up in 3 responses, so would be a good candidate to base my game off. Mario games tend to be simple, but also have tight controls which can add to the challenge later in the game. My previous research also shows that Super Mario World is one of the best 2D platformers of all time, and these responses support this idea. However, it is also important to consider mechanics from the other games mentioned, such as Rayman, Sonic and Celeste.

Q2



Whilst the vote is fairly even, the majority of people have not heard of Celeste before. This is likely because of the game being an Indie game and not very mainstream, combined with the decline in the quantity of 2D platformers since the 16-bit era. Celeste, being a newer game, has not reached the popularity of Mario shown in the previous question, despite having very good reviews that compete with those of Mario games with a much larger production budget and team.

Q3+4

What makes a good 2D platformer for you?

16 responses

- Not too complex
- platforms
- Plenty of interesting mechanics that don't get forgotten immediately after being introduced, but used throughout levels from then onward to create interesting gameplay
- jumping
- Good core mechanic
- good engagement and difficult to pass levels
- Good bosses, storyline
- Level Design: rather than "what do i do", make them think "how do i do this"
- Simple to play 🔥
- Good looking levels(foreground and background art), range of enemies at different skill levels, boss fights, power ups, increasing difficulty as the game goes on
- Good graphics style and fun levels
- lost of fun and enjoyable content
- Good jump
- Tight controls
- Solid jump
- Movement

Are there any particular mechanics you would like to see in the game?

16 responses

jumping

Jumping and Power Ups

Plenty of aerial mobility options (eg. dive, ground pound, something to make moving in the air interesting)

running

Double jump

double jump, 8 way aiming

Wall jumping and double jump

Not really tbh

Power ups, life system, endless mode

Jumping and attacking enemies

Loot boxes

Jump

Jumping

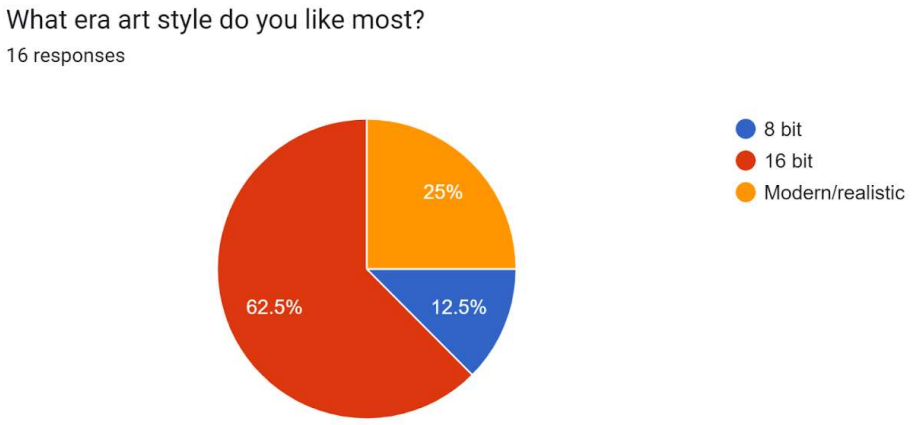
Left and right movement

Tight controls

Many people have mentioned the importance of the jump mechanic; whilst simple, it is clear that people want the basic mechanics to be solid before considering any unique features. People also want tight controls/movement and a good graphics style, which I have asked about in more detail in a later question. I do wish to make the game increase with difficulty as it goes on - a trend followed in pretty much every game regardless of genre. One person also mentioned they don't want a mechanic to be forgotten about once it is introduced, but to be used throughout the levels. I think I will be very careful about what unique mechanics are used and introduced where and how they can be used together to progress through a level. I also will try to make the mechanics fun and not just there to add features unnecessarily. The introduction of power ups will have to be done in moderation, as I would like the main focus to be on the tight controls which many people have asked for, I also want the game to to be a good platforming challenge

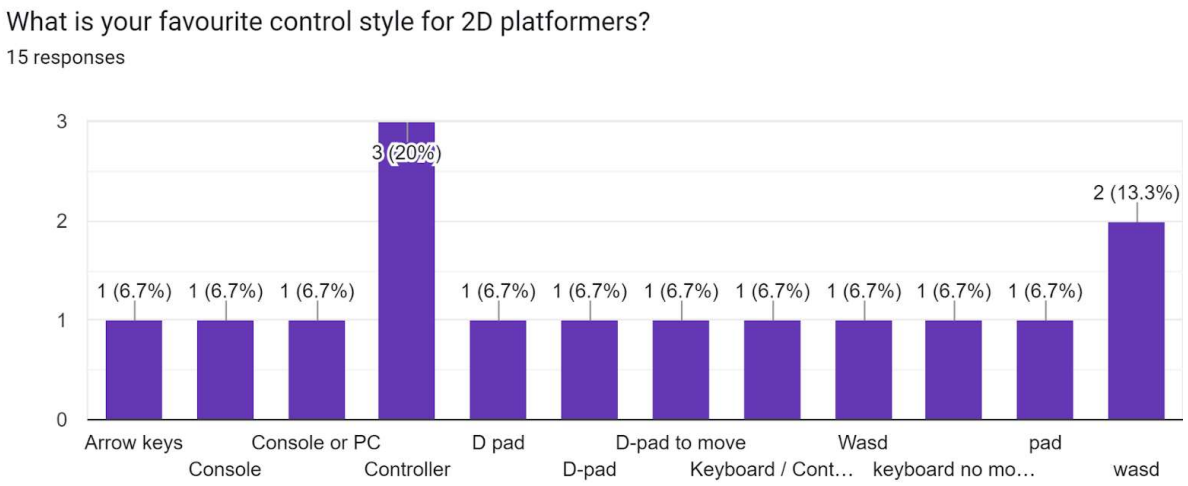
without too much assistance from power ups. I will also not focus too much on enemies, keeping the focus on the skilled platforming and tight controls. Aerial mobility options, boss fights and an endless mode seem potentially a good option, but I am very concerned with the very complex code that would be required. I may consider them as an extra objective, but these will not be a main priority during development.

Q5



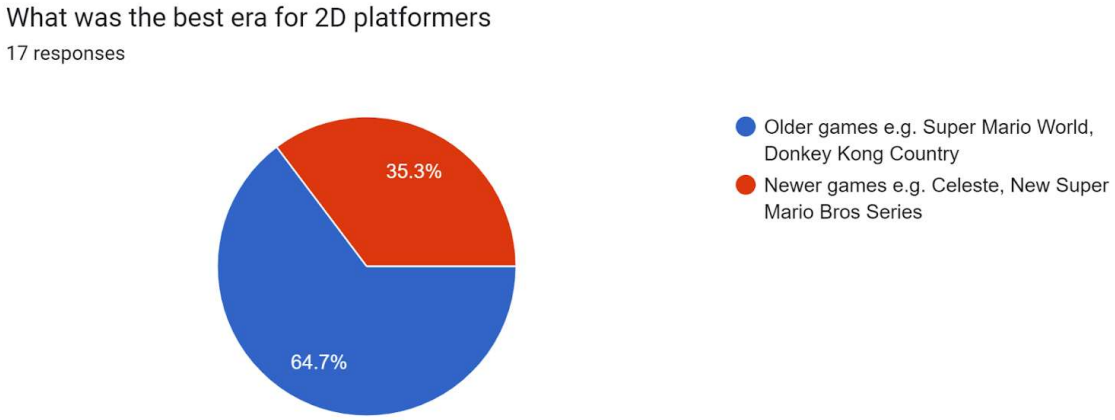
The majority of people prefer the 16-bit art style of games such as Super Mario World and Celeste. Therefore, I will base the art style of my game on this, and choose a 16-bit art style. I plan on using free 16-bit assets from the Unity store in my game, as my art skills are not very good. This will also help me to focus more on the coding and game engine during development. I think people chose 16-bit art style the most because it is more advanced and looks more polished than an 8-bit art style, but is associated with 2D platformers more than a realistic art style; I believe this is because the 16-bit era was considered the best era for the 2D platformer genre because the quantity of good 2D platformers has diminished ever since.

Q6



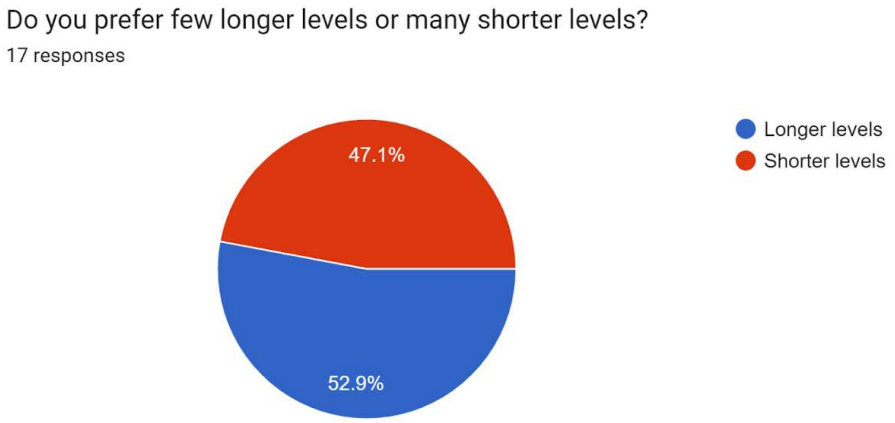
People are split between controller and keyboard for their preferred control style. Whilst I would personally use a controller, specifically the d-pad as many people have suggested, I can see why people would also enjoy using a keyboard to move their character, such as with “WASD” or the arrow keys. Either of these options should allow for tight controls of the player which I wish to achieve. I have no problem with accommodating both of these control options as the Unity input system is very good at allowing multiple input options for the same control input in the game. I will develop the game using “WASD” keyboard inputs during development as I will not always have a controller with me, but I endeavour to support both later on into development.

Q7



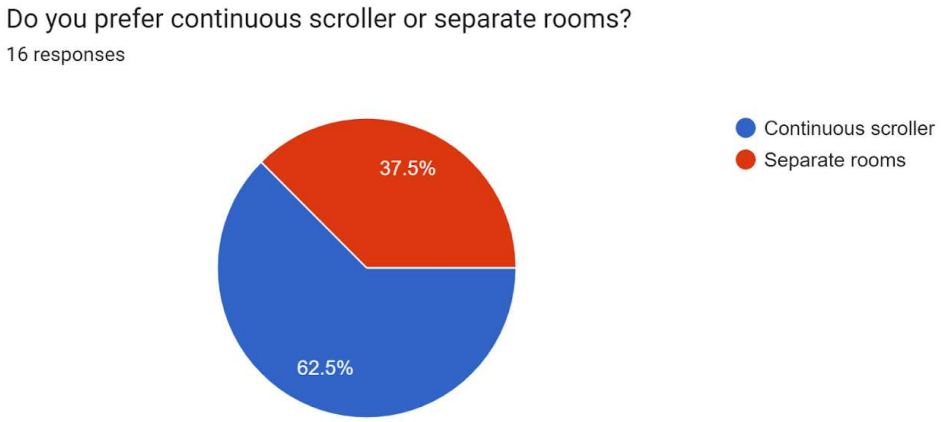
Despite my focus group being 16-17 years old, the majority of people prefer older 2D platformers. I am not surprised given the many answers of Super Mario World to the “What are your favourite 2D platformers of all time?” question. Although, I would have assumed people would have a bias to the newer 2D platformers if they grew up playing them in the early 2010s. This shows just how influential the 2D platformers of the 16-bit era were, even 30 years later. This further shows I should base my game around Super Mario World, arguably pne of ht best 2D platformers of the 16-bit era.

Q8



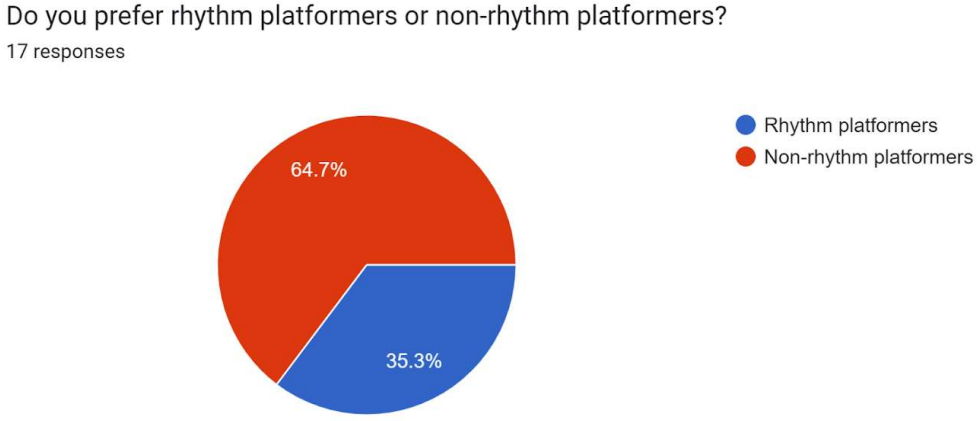
The choice between longer and shorter levels is very split. There is a slight majority of votes for longer levels, but this is not too conclusive. I think this shows that people would be happy with either length of level. This leaves me with lots of options when it comes to level length. I could have either type, or even a variety of level lengths. I am considering using longer levels, making each level more distinct, potentially with its own unique art style and/or mechanics. I feel this could be a good way to make sure each mechanic is used properly and is not just forgotten about soon after it is introduced. However, this is not final and may change later during the analysis, design and the actual production stage. Unity allows the developer to have multiple “scenes” which is very helpful for making multiple separate levels.

Q9



The majority of people prefer a continuous scholar, similar to that of Mario Games. I think this will be easier to code than separate rooms. I think people chose continuous scroller more because it gives a more open feel, making the player feel they have more control over what they do. Being that there was nearly 40% of votes for separate rooms, I could add some - still keeping the main camera system as continuous scroller - but this would be quite challenging, so would have to be an extra, optional objective that I could potentially implement at the end of development. Continuous scroller is more common in the 2D platformer genre; some games such as Rayman Origins/Legends and Celeste use a combination of the two, and Mario games tend to have some separate bonus rooms with ‘Star coins’ (special items) in them.

Q10



The majority of people said they prefer non-rhythm platformers. This means the progression of the level is not tied to particular music. Therefore, the music will be background music, not

designed to influence the player too much if at all. I also think that music levels such as those in Geometry Dash and Rayman Legends would be too time consuming and difficult to implement, so I will definitely be sticking to non-rythm level design. I also want the player to be able to progress through the level at their own pace, not dictated by a music's tempo.

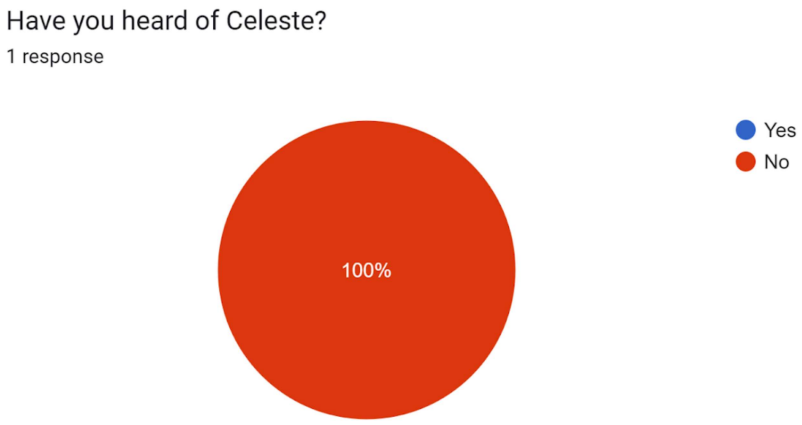
Questionnaire with client:

My dad likes 2D platformers but he no longer has any to play due to a lack of new games of this genre. The number of 2D platformers has declined since the end of the 16 bit era, so he is running out of games to play.

Q1

This further shows how much of an impact Super Mario World has on the 2D platformer genre despite being over 30 years old, hence I will further base my game on Super Mario World as the main inspiration. Particularly, I want my game to have tight controls and a similar art style to Super Mario World.

Q2



This suggests that older generations have not heard of Celeste as much, potentially because it is much newer compared to older games such as Super Mario World and because it is an indie

game with a lower development and marketing budget. However, I do think that my dad would enjoy Celeste if he played it, due to its unique dash mechanic and 16-bit art style. I will also partly base my game off Celeste and use it for inspiration.

Q3+4

What makes a good 2D platformer for you?

1 response

Sensible side scrolling screen with floating/moving platforms to move onto. Jeopardy, time deadline.

Are there any particular mechanics you would like to see in the game?

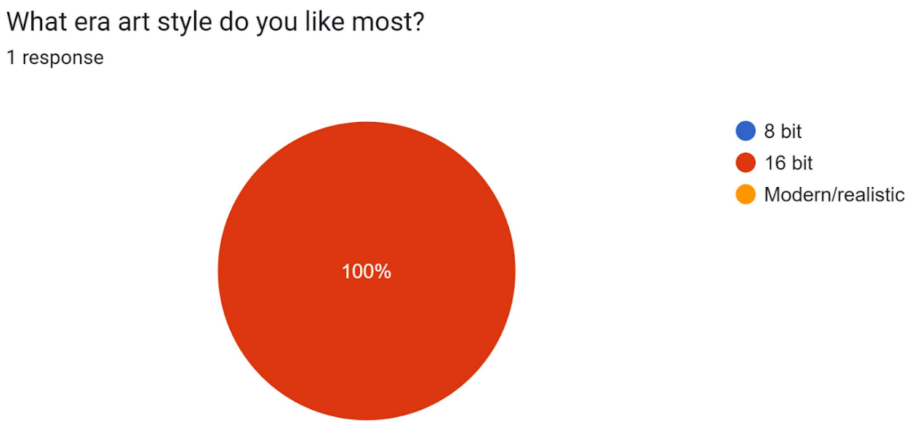
1 response

Clouds, platforms that drop after you've landed on them, pendulums or falling ceilings, landposts to bookmark progress.

The game will definitely be a scroller and not just rooms. The moving platforms are a must and I already have some ideas about waypoints to direct the platforms to move towards. I will consider the time deadline and hopefully make a way for times to be recorded per level/the whole game to further add to the jeopardy.

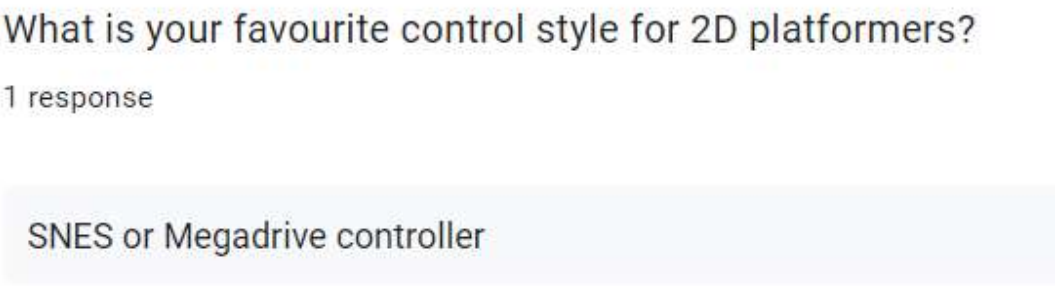
I will try to include as many of these mechanics if they are technically viable, because they can make the game more fun, but may be hard to implement in code. Falling platforms would add to the time deadline, rushing the player and adding more difficulty to the game. I will definitely have the landposts at the end of the level, and potentially throughout the level (such as a half-way point) if technically viable.

Q5



I will make the art-style of my game 16-bit. I will find 16-bit assets from the Unity store and import it into Unity, which has many options for 16-bit sprites and tile pallets for terrain. 16-bit assets have to be set to 16 pixels per unit (units are squares on the grid in the scene view in Unity). The terrain and background assets can be added to a tilemap to paint the tiles onto the scene, with terrain and background on separate layers.

Q6

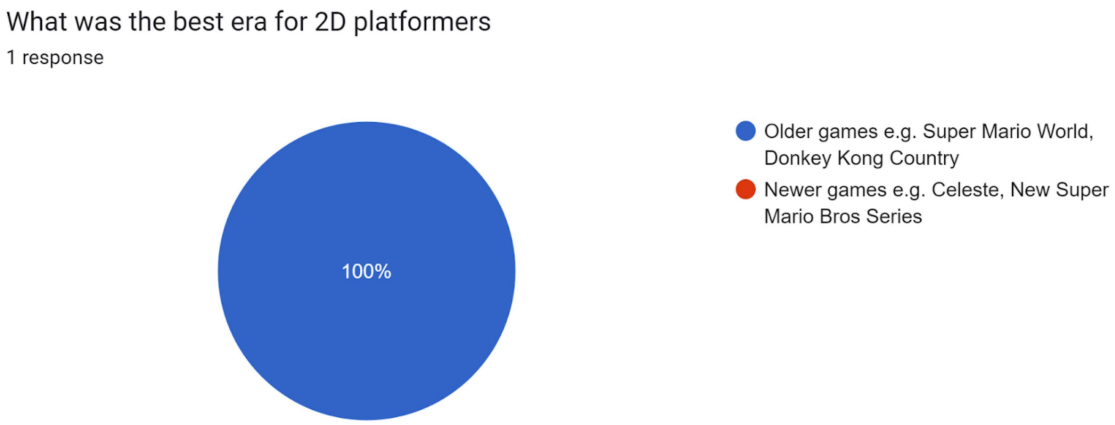


Both the SNES and Megadrive controllers used the d-pad to move the player. Therefore, I will have d-pad support, but potentially also add support for joystick and keyboard input. This would be possible with the new Unity input system. I will test my game with a controller I have[16] that is new, but is heavily based off the SNES controller, just with added joysticks.

---

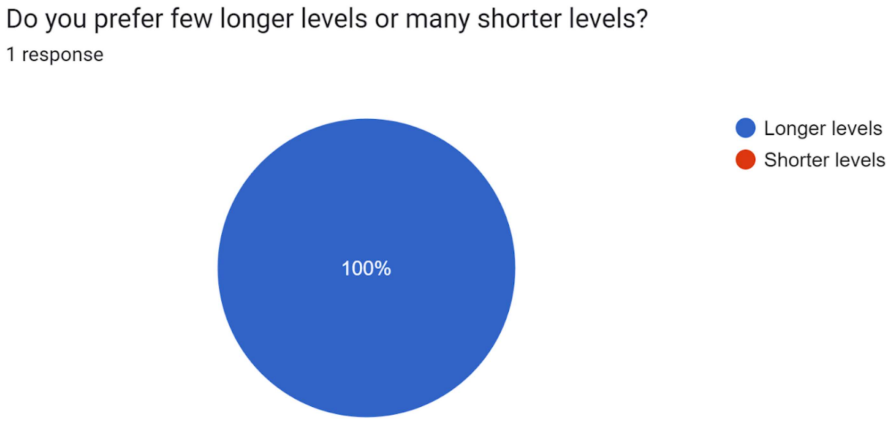
[16] Controller
https://www.amazon.co.uk/8Bitdo-Bluetooth-Controller-Android-Raspberry/dp/B08XY8SK9B/ref=sr_1_9?crid=UV5TUU4IW5NU&dib=eyJ2IjoiMSJ9.s3t3QVn06vnN-_jeSv6WGRcMnZ_6n0hgGZhz0vg4Ep6Zorp-uDOkcxhVB8-3zqms2shnsehK-0YXju2COQWHrKE89L-BT75L-dQ54pf6VmpvI0ecDSPuBJBK1fzi2t4fmd2gKidfiPgmZOvIm3L3XKgzO3Wg8C6A57IXh0rZwrG0ZsEotquL1cgRQ5t9tVAcVt23jHDncYdkX2enTzEEI2z0RQPq5gy9WFqBqpJqhD8.5CH6J-EQq7NZEaPtRb3I0VZRxUVZypI6uNeKTxT5C58&dib_tag=se&keywords=8bitdo%2Bcontroller&qid=1720772806&sprefix=8bitdo%2Bcontroller%2Caps%2C163&sr=8-9&th=1
Accessed 12/07/24

Q7



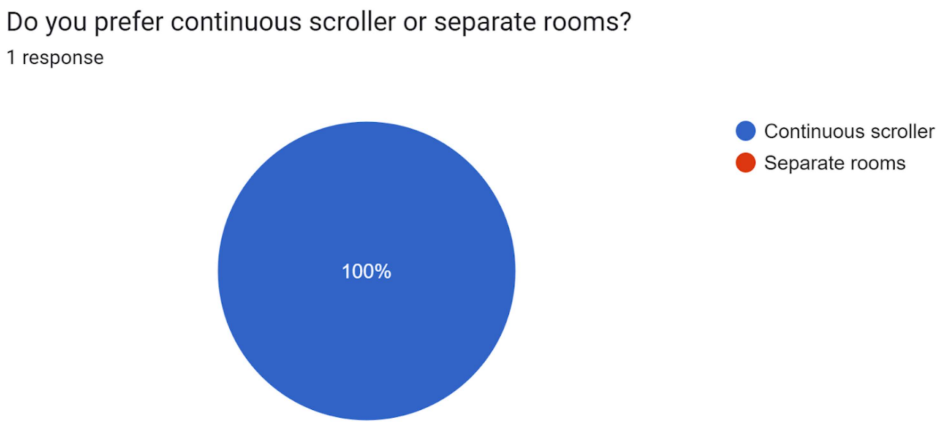
I will base my game off older style 2D platformers, primarily Super Mario World, but may take features from newer games if I see fit. The focus on older games will appeal very well to my target market.

Q8



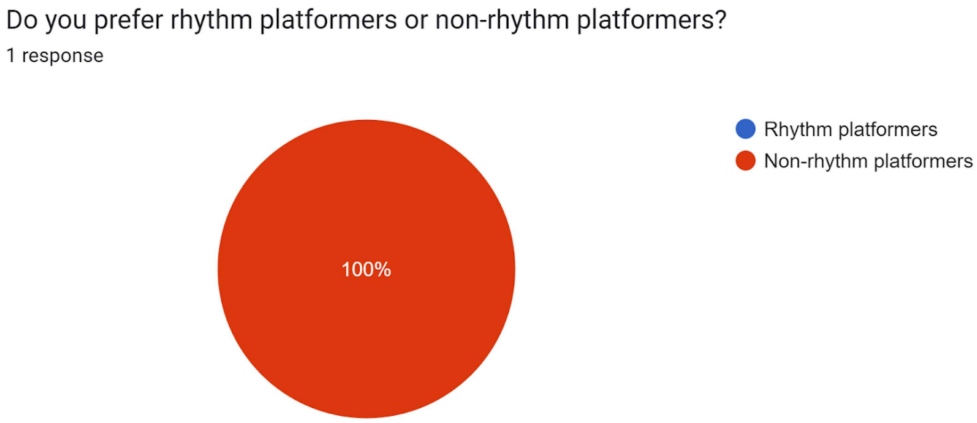
I am happy to make fewer, longer levels. This would make each level feel unique which would make the game feel more enjoyable to play. This also gives enough space in the level so will give a level its own unique mechanic and colour theme that is not present in other levels. I will aim for a longer level length similar to those in Rayman games, rather than the short/medium length levels of Mario games.

Q9



Each level will be one continuous scroller. This gives the player more freedom to complete the level how they choose to. It also gives a focus more on the platforming than a problem solving element, although a problem solving element could still be implemented secondarily. I do plan to have mostly horizontal scrolling but some vertical scrolling, to make certain levels unique, with that being its unique mechanic. This also allows the player to move through the level at their own pace, without having to wait/pause their speed when changing rooms.

Q10



The game will be a non-rhythm platformer, which will be easier to design and be closer to Super Mario World, but I will still have music in the background (alongside sound effects) to add to the feel of the game. The first levels would have more upbeat music, but potentially later levels would have more scary music to build suspense and add pressure to the player. Also, the tempo of the music could increase when time is about to run out, should I add a time limit feature to the game. A game without music would feel incomplete.

Playing previous games

As part of my research I have played a range of 2D platformers such as Super Mario World, Rayman Legends and Celeste to refresh my knowledge of 2D platformers; these are the games I wish to take the most inspiration from when designing and developing my game . This has given me a good idea of level length (I wish to make my level length similar to that of Rayman games). I really like the 16-bit art style of Super Mario World and Celeste and I will apply a similar art style to my game. I will continue to play through Super Mario World (as seen in the photo) as I develop and test the game as it is the game I will be basing my game on the most. I particularly want to implement a similar control style in my game. The original console, controller and CRT tv really help me to experience the game as intended, and I want my game to feel as good to play as Super Mario World does. The rewind feature when playing NES Mario games has proved very useful, and I plan on developing my own similar feature so I can have this functionality in my game.

Objectives

Some objectives are more easy and quick to complete such as object-oriented programming, traps killing the player and an individual level timer. These are essential to the game and will be hit earlier on during development. However, some are more aspirational, complex and time consuming objectives, such as the recall system, API import and file saving. These would be nice to hit if development time allows, and I will aim to hit these, but some may need to be cut if I am low on time or the features are too complex.

Technical Objectives

- Use the Unity game engine and C# programming language
- Implement object oriented programming, including classes and inheritance
- Implement stack operations to allow player to set/recall their position
- Implement list operations for:
    - Waypoint following
    - Times to beat levels
    - Quotes used if API import fails
- Implement file saving (text file and/or SQL) for direct access to save:
    - Date of completion
    - Times for each level
    - Total time
    - Times sorted using merge sort
- Utilise a merge sort to sort level times
- Utilise a recursive algorithm in the merge sort
- Import a web service API and use JSON parsing to give the player a message for completing the game
- Use some form of complex maths
- Use vectors for
    - Player movement
    - Platform movement
    - Trap movement
    - Enemy movement
- Animate game objects
    - Player has movement states for:
        - Running
        - Jumping
        - Falling
        - Idle
        - Death
    - Basic animations for:
        - Enemies (spike turtle more advanced)
        - Traps
        - Platforms
        - Power ups

Gameplay Objectives

- Game meets requirements of client
- Camera follows the player’s position
- 16 bit art style, similar to that of Super Mario World
- Start screen
- Main levels

    - Multiple levels
    - Varying length
    - Horizontal vertical scrolling
    - Levels in approximate difficulty order
- End screen with random quote from API
- Allows keyboard, mouse and controller inputs
- Recall system
    - Spawn point can be set (pushed onto stacks)
    - Player can teleport to the last spawn point (pop from stack)
    - Audio plays on spawn set and recall triggered
    - Only pops from stack if it is empty
- Time how long it takes the user to complete:
    - Individual levels
    - The entire game
- Track number of fruits collected throughout the game
- UI
    - Persistent throughout levels
    - Displays:
        - Number of fruits collected (integer)
        - Total/global time (float to 2 decimal places)
        - Individual level time (float to 2 decimals places)
        - Time to beat each individual level (list of unrounded floats)
- Audio
    - Backgrounds music changes for different levels
    - Player audio
        - Jumping
        - Death
        - Recall point set
        - Recall triggered
- Pause menu
    - Buttons to:
        - Resume game
        - Quit game
        - Return to start screen
        - Skip level
    - Timers pause
    - Gameplay pauses
    - Background dims
- File for direct access contains:
    - Date of completion
    - Times for each level
    - Total time
    - Total fruits
    - Times sorted using merge sort

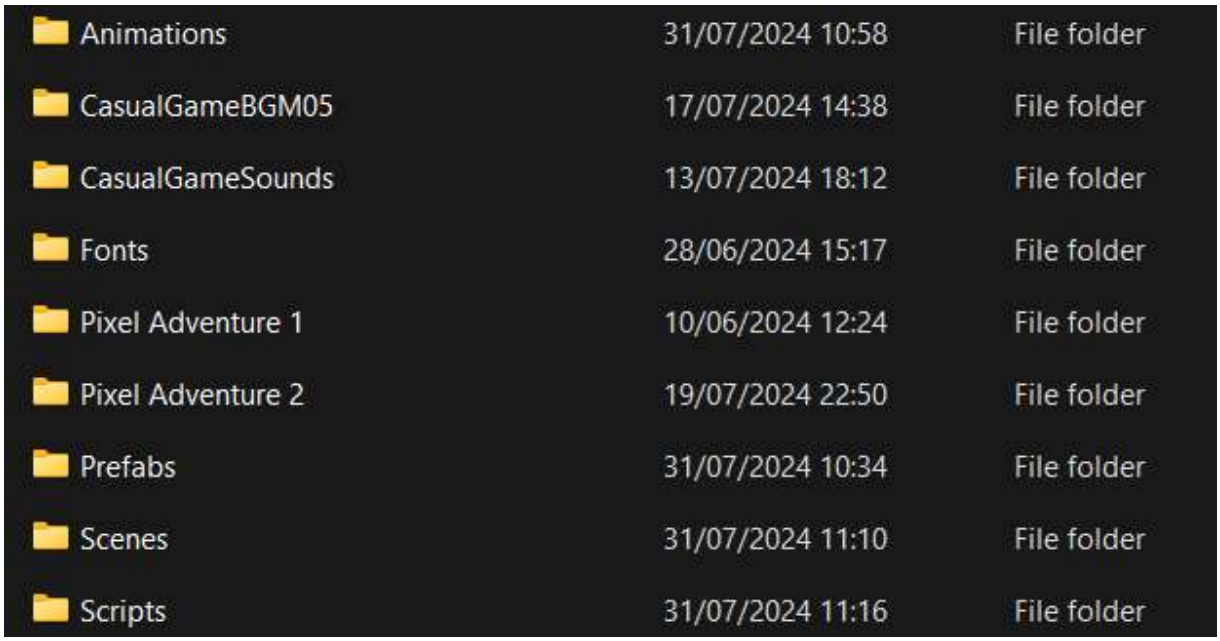
Level Design Objectives

- Enemies/traps
    - Kill the player
    - Can move on paths
    - Flying enemies can:
        - Fly between waypoints
        - Vary their height
        - Adjust velocity
    - Slime can be killed if jumped on, triggering a death animation
    - Spike turtles
        - Change between being a trap or not
        - Change between being normal ground or not
        - Spikes come in and out with animation
    - Falling spike head
        - Falls when player walks beneath it (trigger box)
        - Can adjust speed of falling
    - Rotating spike balls can have their rate of rotation and direction adjusted
    - Flame kills a player if it touches it
- Platforms
    - Swinging platforms
        - Can adjust speed of swing
        - Can adjust direction of swing
        - Can offset the swing of one platform from another
    - Moving platforms
        - Can be jumped on
        - Carry player with them
        - Can adjust movement speed
    - Disappearing/recurring platform
        - After a certain time become dim and cannot be stood on
        - Can change time for which platforms are active
        - Can change time for platforms to respawn
    - Rotating platforms
        - Platforms follow end chains
        - Can change speed and direction of rotation
    - On/off platforms
        - Change between active and passive by becoming dim or not dim and being able to be stood and jumped on or not
        - Can adjust time before changing state
        - Alternation can be offset between alternating platforms
- Flying item takes inputs from the player when the player jumps into it
- Finish flag loads next level after waiting x seconds
- Trampolines/fans launch player up
- Arrows show general direction of level progression

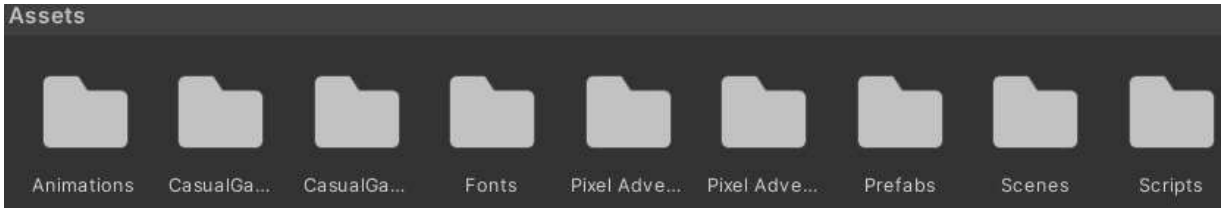
Documented Design

File organisation

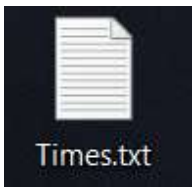
To enable my project to be maintainable, it will be important to organise files into distinct folders with sensible names so that files can be located easily during development. Without this, there would be many files all held in the root folder. This organisation should ensure I do not waste time searching for files and helps me to not lose any files that I would have to remake from scratch; this will make development more efficient so I have more time for creating the game itself. The root folder accessed during development is the “Assets” folder. Within this root folder I will have folders for various types of assets such as scripts, animations and prefabs (created game objects saved so they can be added in other parts of the game). My art assets will also be stored here (see next section) and possibly save files with game progress and/or completion time, should I add this feature. Times will be saved on the desktop so they can be accessed easily



Early assets folder seen from file explorer



Early assets folder seen from Unity#



Times.txt file on the desktop

Game art and music

I will import my game art and music from the Unity Asset Store[17], because I do not have adequate art skills to create my own assets. This will also enable me to spend more time on the coding and level design, rather than the time consuming process of art creation. I will import the assets used to the root assets folder. For art I will use “Pixel adventure 1[18]” and “Pixel adventure 2[19]”. For background music I will use “Casual Game BGM[20]” and for sound effects I will use “Casual Game SFX Pack[21]”. All of these assets are free. Sprites must have their “pixels per unit” value changed from 100 to 16, and terrain and background art must be sliced into 16x16 tiles.

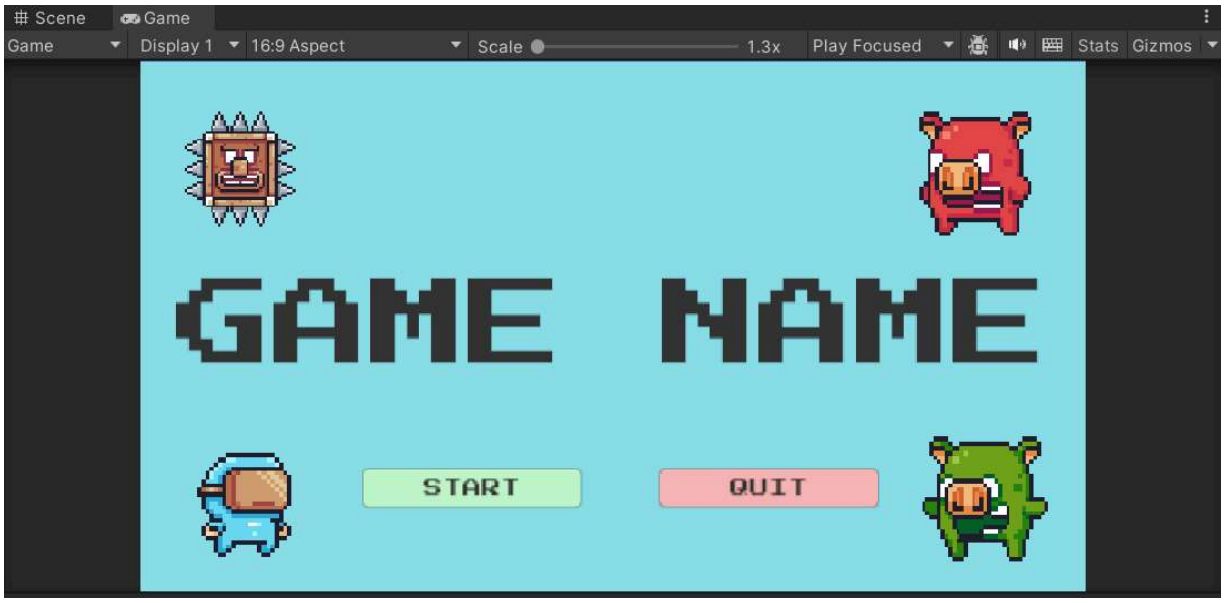
User Interface

The game has a graphical interface which allows the user to interact with the game. I will have a start and end screen alongside regular gameplay. If time allows, I would also like to add an options menu. To begin with, the menu design will be limited, but I will improve how the menus look if I have time closer to the end of development.

[17] Unity Asset Store https://assetstore.unity.com/ Accessed 03/08/24
[18] Pixel adventure 1 https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360 Accessed 03/08/2024
[19] Pixel adventure 2 https://assetstore.unity.com/packages/2d/characters/pixel-adventure-2-155418 Accessed 03/08/2024
[20] Casual Game BGM https://assetstore.unity.com/packages/audio/music/casual-game-bgm-5-135943 Accessed 03/08/24
[21] Casual Game SFX Pack https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116 Accessed 03/08/24

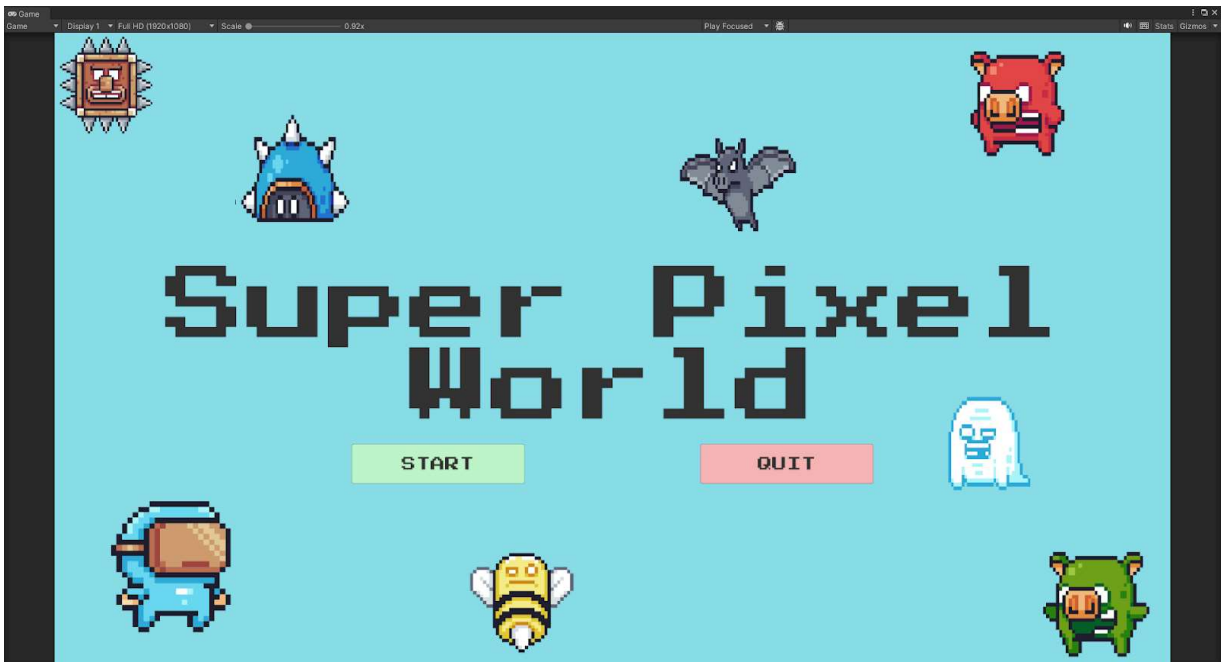
Start screen

Early version of the start screen.



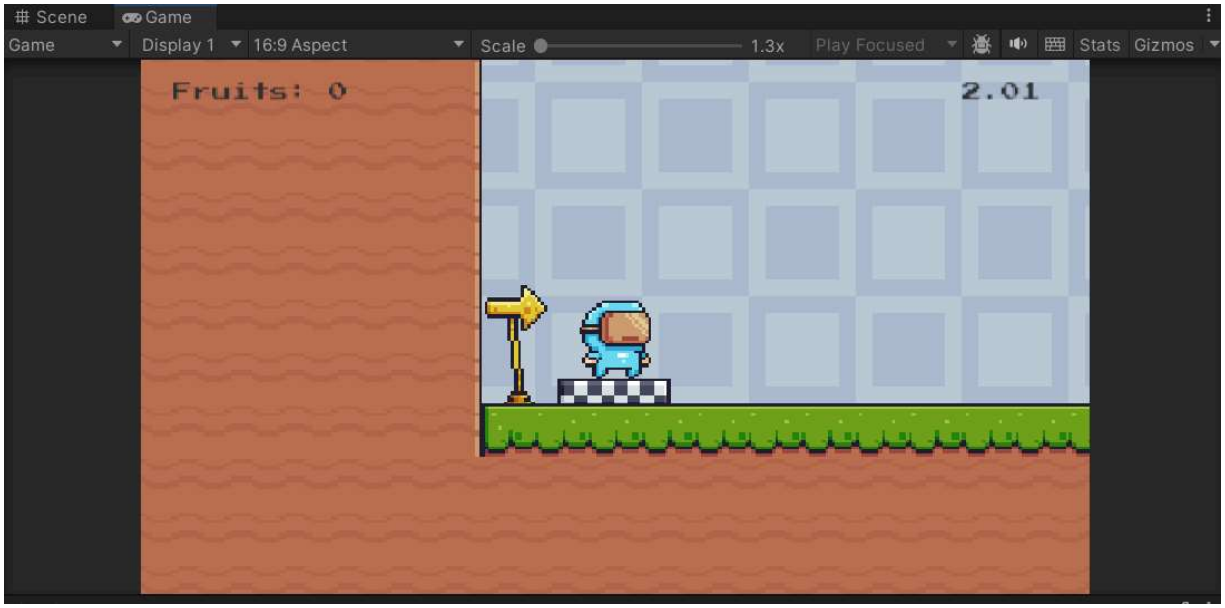
Start menu mid development.

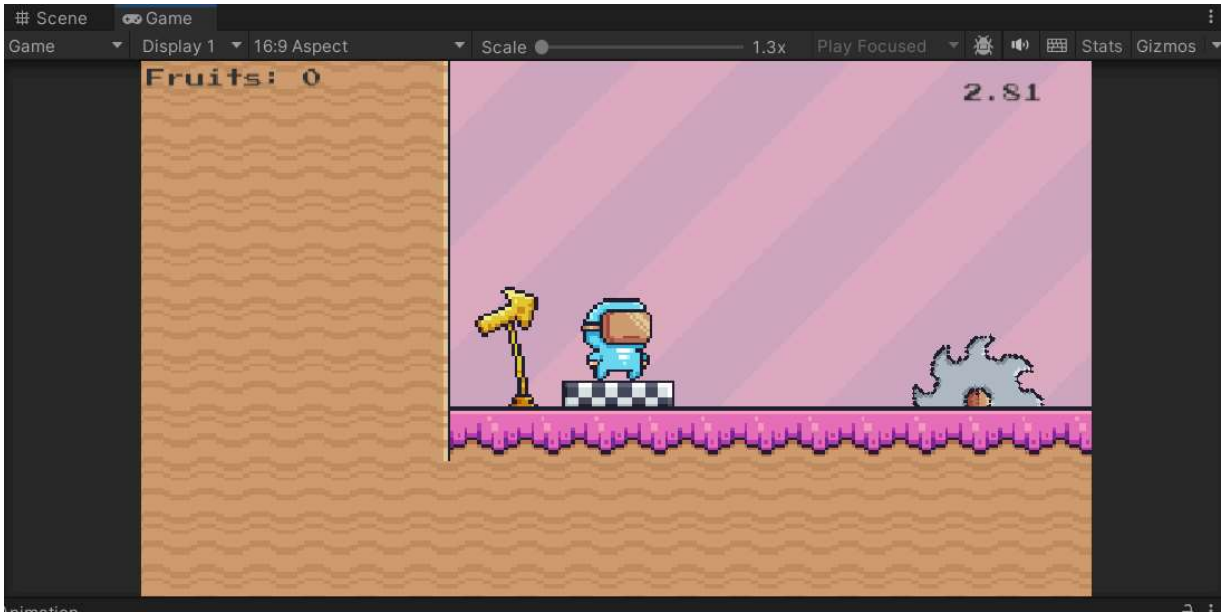
“GAME NAME” will be changed to the actual game name later once I choose what the game will be called. The start button is a dimmer green whilst the mouse is not hovering over it, but when the user hovers the mouse over it, the button changes to a brighter green as seen in the early version above. The start button begins the game by changing to scene 1 (with the start screen being scene 0). I will add more design to this menu at a later stage of development with more art, but at the moment it is only the idle player sprite from the start of the idle player animation.



Game name added and scaled to full 1920x1080.

Gameplay





Each level will have its own colour theme. Above is level 1 with a green/grass theme and level 2 with a pink theme at the time of writing. The player is always at this position in the camera, slightly below centre, so that obstacles can be seen well enough. There is currently a fruit counter and timer on screen in the top left and right corners. Should I add any more HUD items - such as potentially a death counter - towards the end of the development, I can add them to the bottom left and right corners, and/or along the top. All HUD items shouldn't obstruct the main gameplay, so the user isn't distracted by them and can always see obstacles as much as possible.

Final UI design. Fruits counter top left, global and local timers top right to 2 decimal places for easy reading and times for levels 1-5 respectively bottom left to 6 decimal places to be more accurate.

End screen

Shown is an early end screen of the game. I will likely add more art/design towards the end of development should I have time. The buttons change from a dimmer colour (as seen in the quit button on the right) to a brighter colour (as seen in the restart button on the left) when the user hovers over them with the mouse. The restart button currently sets the scene to scene 1 in the scene manager (level 1) and the quit button quits the game entirely.

Scaled to full 1920x1080 aspect ratio.

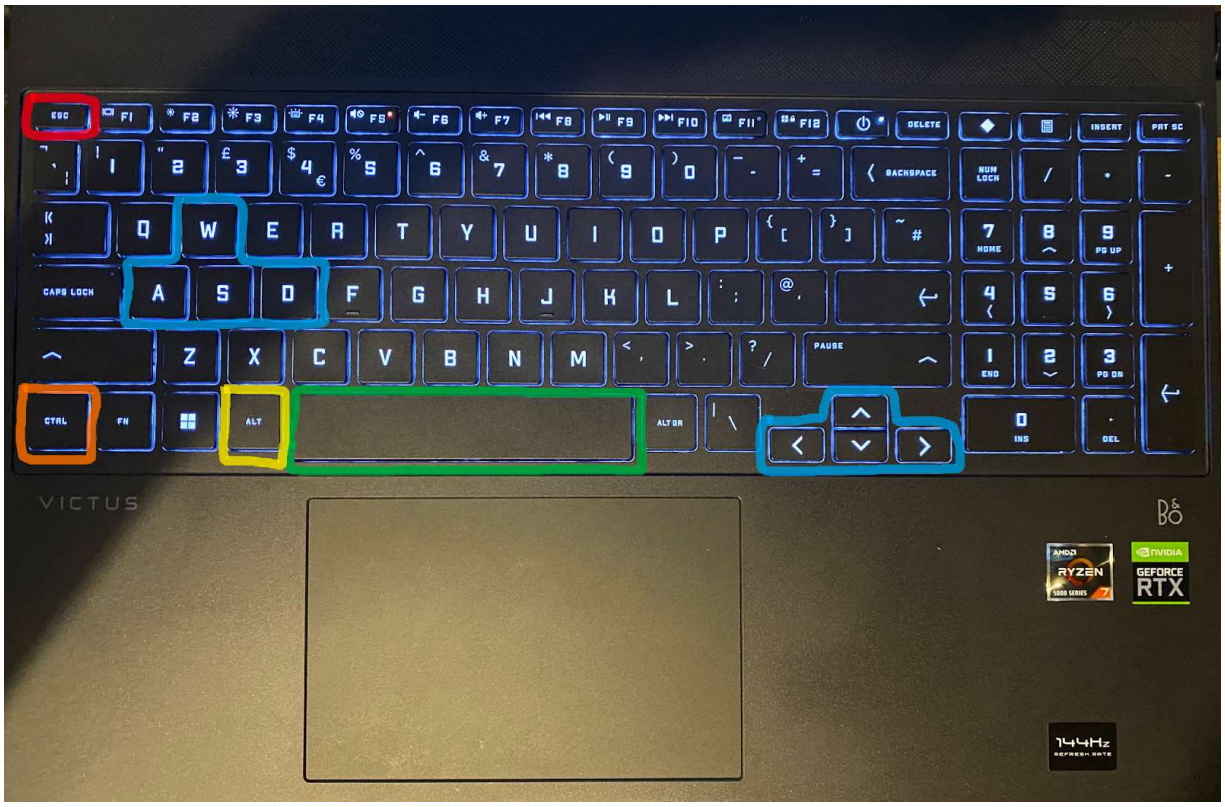
Added a random quote imported from an API (or a random quote from a predefined array if the API import fails).

Inputs

Input devices

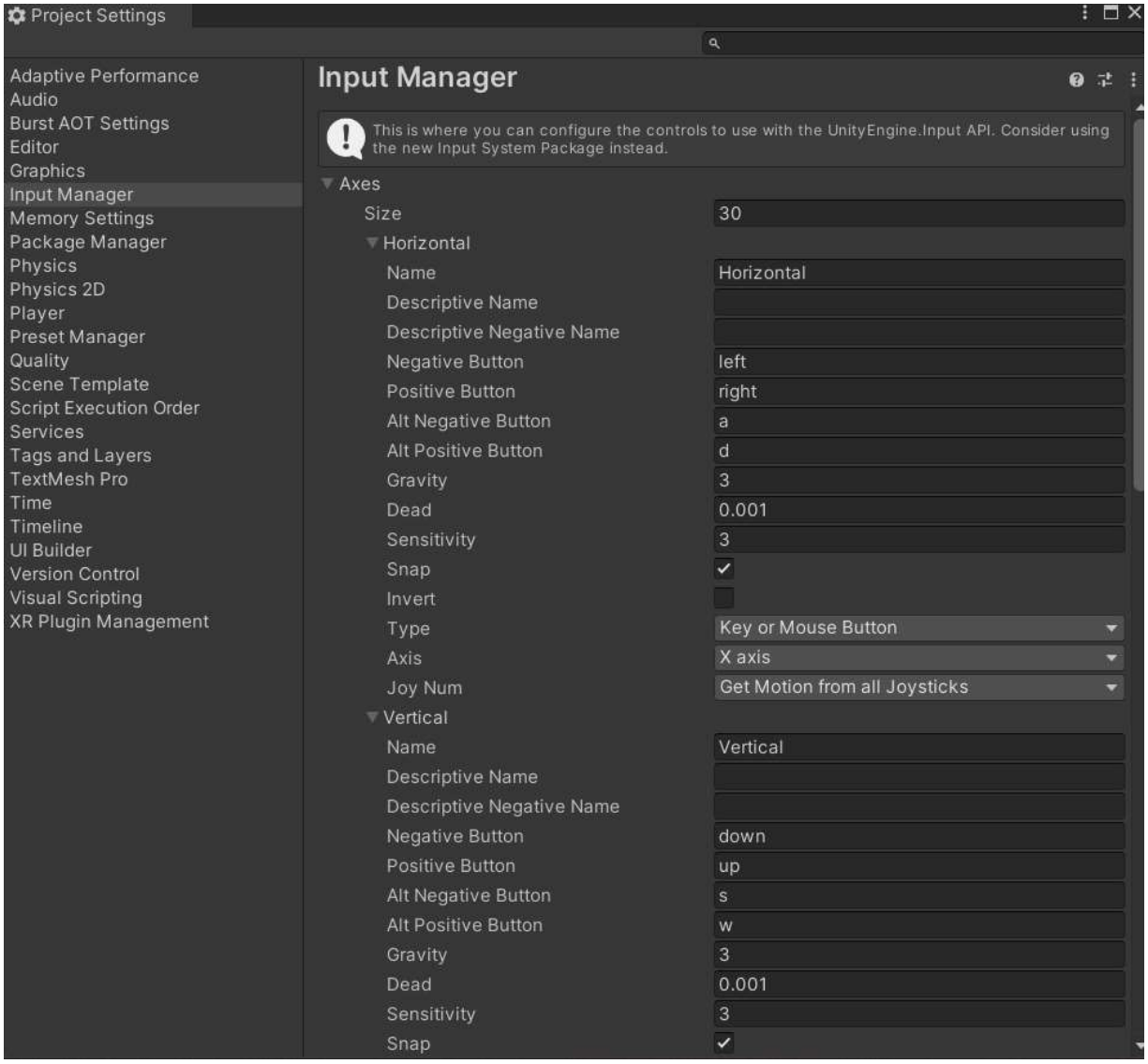
The player can be controlled using either a keyboard and/or a controller. Menus are navigated using a mouse. Exact controls are as followed:

- Movement
- Jump
- Pause/unpause
- Set recall point
- Trigger recall/teleport

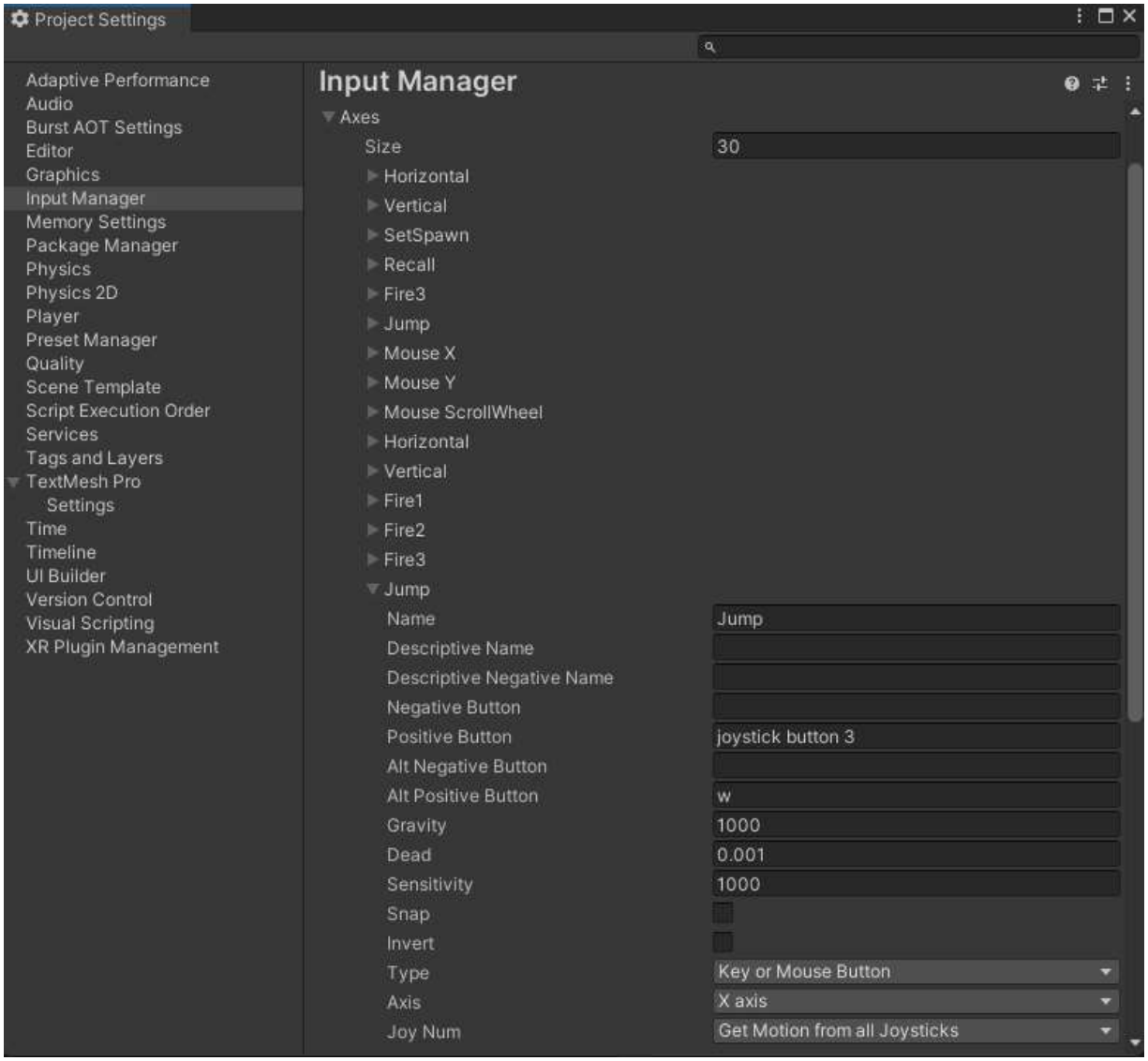


Unity Input System

I will use the Unity Input System to take inputs from the player. The Unity Input System has many in game actions that can be assigned any button in the Unity engine. During development I will mostly have this bound to keyboard buttons as I will not always have a controller with me while I make the game, but will ensure I add controller functionality by the end of development as outlined in my analysis.



Unity Input System at start of development.



Added controller support and additional input types “SetSpawn” and “Recall” for the position recall system.

Components

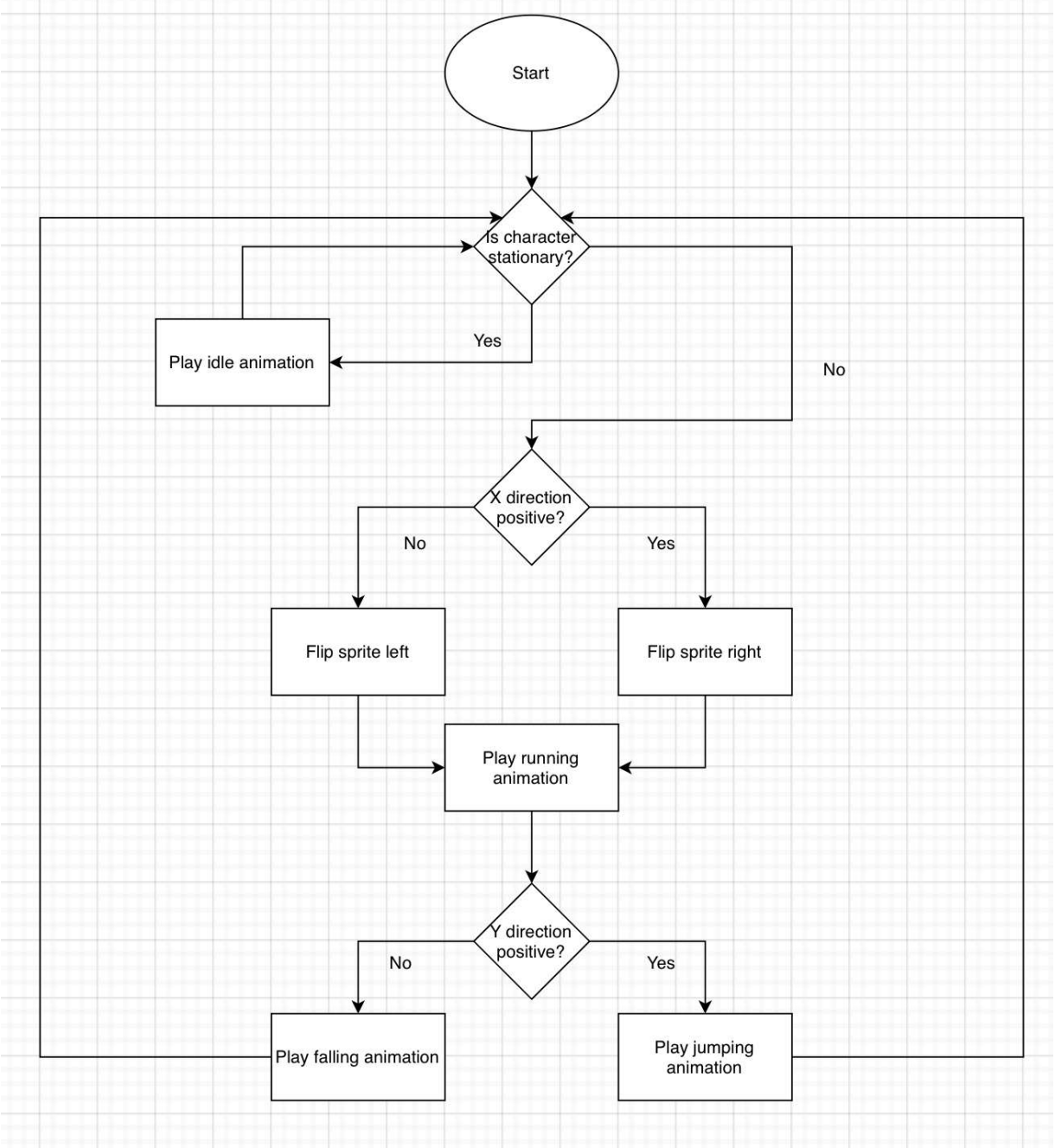
Unity premade components

The player will need a sprite renderer to be displayed on screen, Rigidbody 2D for physics functionality, a Boxcollider2D for collisions, audio sources for sound effects and animator to use animations

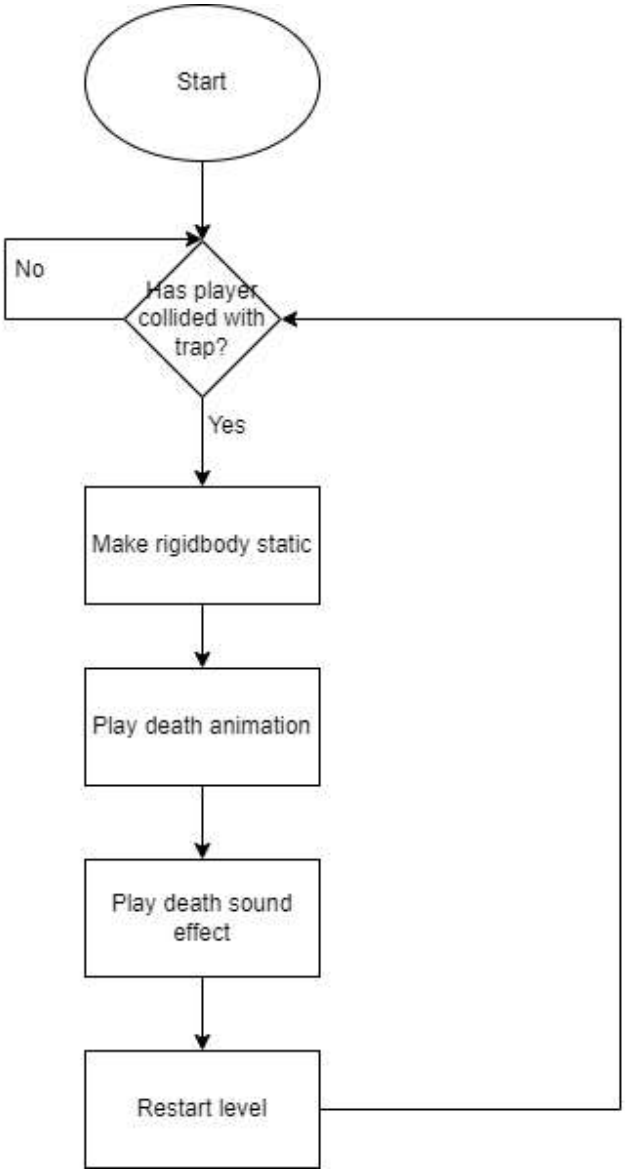
C# scripts

Many features of the game will be added in C# scripts so I can have more control over how the game works and how the user interacts with the game. I will have data flow diagrams and/or pseudocode to design these.

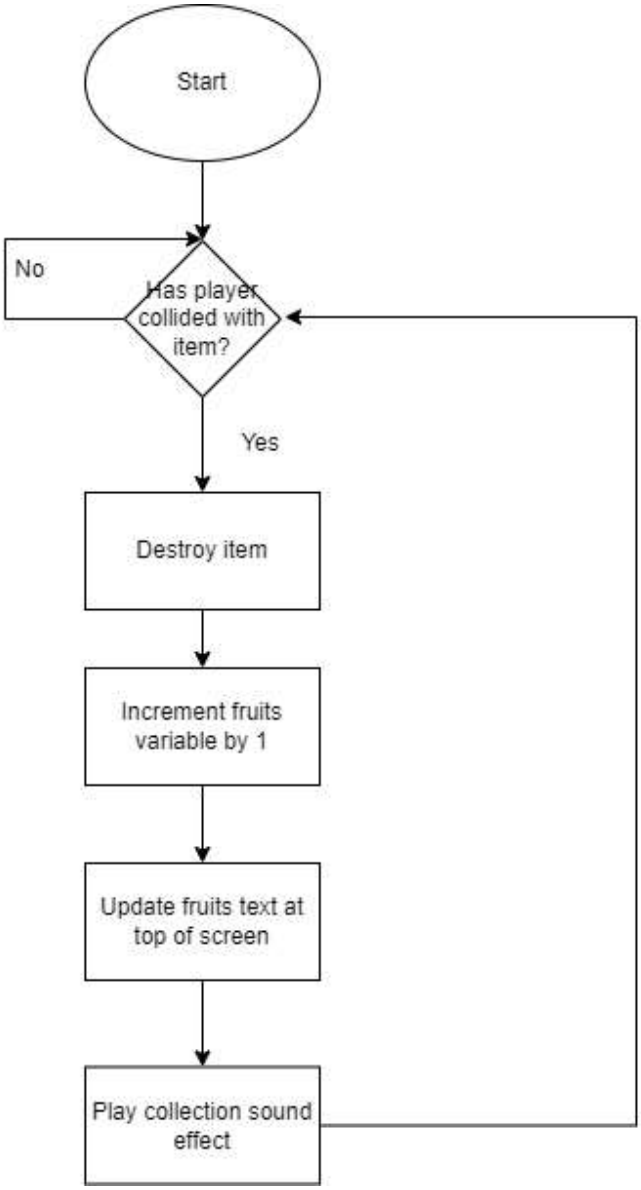
Player movement script



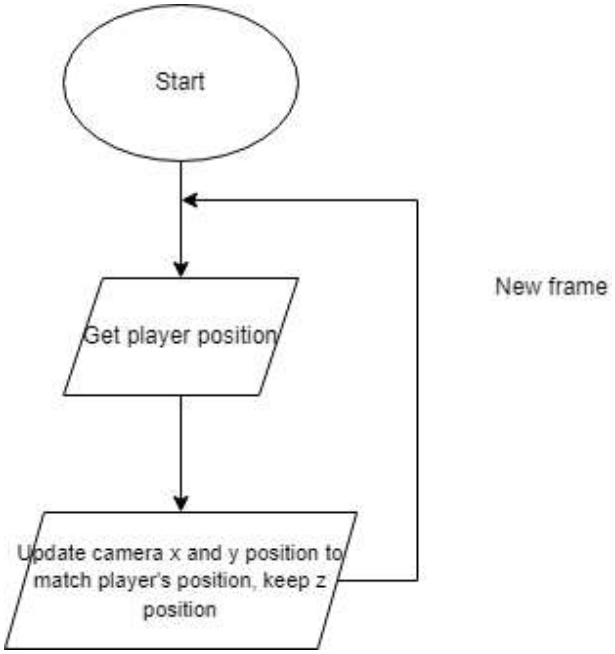
Player life script



Item Collector script



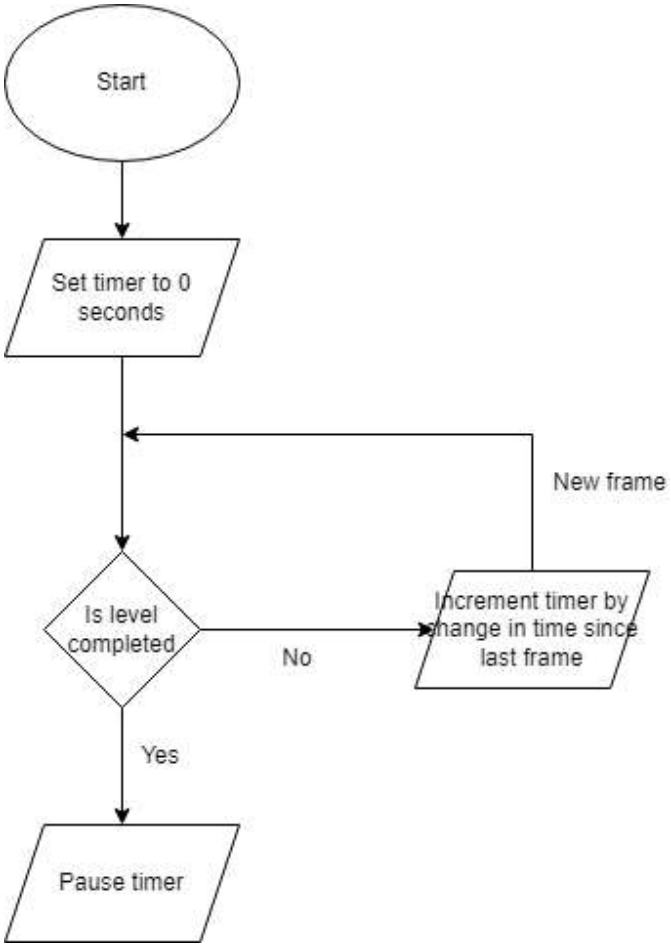
Camera controller




```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    [SerializeField] private Transform player;


    // Update is called once per frame
    private void Update()
    {
        //Each frame the camera is transformed to same position as player, keeping different Z value so the player can still be seen
        transform.position = new Vector3(player.position.x, player.position.y, transform.position.z);
    }
}
```

Each frame, the camera's x and y position is transformed to the same as the player. The Z value is kept constant so that the player is always in front of the camera in the z axis.

Timer



At this stage the timer does not need to be reset to 0 after the level is completed because this script is run separately for each level.

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports Unity UI
using UnityEngine.UI;

public class Timer : MonoBehaviour
{

    [SerializeField] private Text timerText;

    private float currentTime = 0f;
```

```
    private bool TimeEnd = false;

    [SerializeField] private Finish finishScript;


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

    }

    // Update is called once per frame
    private void Update()
    {
        //Only increments if level is not complete (checks using public
variable from finish script)
        TimeEnd = finishScript.levelCompleted;
        if (TimeEnd == false)
        {
            //Increments with change in time since last frame
            currentTime += Time.deltaTime;
            //Updates timer text to 2 decimal places
            timerText.text = currentTime.ToString("0.00");
        }
    }
}
```

First timer manager code **(not final)**. Timer increments by Time.deltaTime() (change in time between each frame) every frame. The timer text on the canvas updates to the new time. Timer stops when finish is detected from finish script (added in serialised field). Timer currently does not continue through levels (starting from 0 at the start of each level) but this could be added at a later date if I have time.

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports Unity UI
using UnityEngine.UI;
//Imports SceneManagement from the Unity engine to allow us to check if game
compelete screen is reached
using UnityEngine.SceneManagement;
using System;
```

```
//Allows writing to text files
using System.IO;

public class Timer : MonoBehaviour
{
    //Allow timer texts to be added in Unity editor
    //Total game time is public becuase it is continued throughout levels
    public Text timerText;
    [SerializeField] private Text localTimeText;
    [SerializeField] private Text arrayText;

    //Float values for time
    private float currentTime = 0f;
    private float localTime = 0f;

    private bool TimeEnd = false;
    private bool skipped = false;

    public Finish finishScript;
    public PauseMenu pauseMenu;

    private float[] times = new float[5];
    private float [] sortedTimes = new float[5];
    private bool written = false;
    private bool fileWritten = false;
    private CollectablesManager collectablesManager;

    private void Update()
    {


        //Writes times to Timer text file and resets timer when the last level
is completed
        if ((SceneManager.GetActiveScene().buildIndex) ==
(SceneManager.sceneCountInBuildSettings - 1))
        {
            SaveTimesToFile();

            //Resets local and global timers to 0f in case the game is
restarted
            currentTime = 0f;
            localTime = 0f;
        }
        else
```

```
        {
            //Resets fileWritten bool when game is restarted from end screen
            fileWritten = false;
            //Gets boolean values for the finish being triggered or the level
being skipped
            TimeEnd = Finish.levelCompleted;
            skipped = PauseMenu.levelSkipped;

            //Only increments if level is not complete and level has not been
skipped
            if (!TimeEnd && !skipped)
            {
                IncrementTime();
                written = false;
            }
            else
            {
                TimeToArray();
                //Resets local timer to 0.00 aftfer level is completed/skipped
                localTime = 0f;
                localTimeText.text = localTime.ToString("0.00");
            }

        }

    }

    private void SaveTimesToFile()
    {
        //fileWritten bool ensures file is only appended to once per game
completion
        if (!fileWritten)
        {
            //Adds line
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**************************************************************************
**********************" + Environment.NewLine);
            //Appends todays date
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
DateTime.Today.ToShortDateString() + Environment.NewLine);

            //Appends each level number and time by iterating with a for loop
            for (int i = 0; i < 5; i++)
            {
```

```
                File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"Level " + (i + 1).ToString() + ":   " + times[i].ToString() +
Environment.NewLine);
            }

            //Appends total (global) time
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Total
time:   " + currentTime.ToString() + Environment.NewLine);

            //
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times
for levels 1-5 respectively:   " + string.Join(", ", times) +
Environment.NewLine);
            //
            sortedTimes = MergeSort(times);

            //
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times
sorted by shortest to longest:   " + string.Join(", ", sortedTimes) +
Environment.NewLine);

            //Adds line
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**************************************************************************
**********************" + Environment.NewLine);
            //Ensures file is not appended to again
            fileWritten = true;
        }
    }
    private void IncrementTime()
    {
        //Increments global timer with change in time since last frame
        currentTime += Time.deltaTime;
        //Updates global timer text to 2 decimal places
        timerText.text = currentTime.ToString("0.00");

        //Increments local timer with change in time since last frame
        localTime += Time.deltaTime;
        //Updates local timer text to 2 decimal places
        localTimeText.text = localTime.ToString("0.00");
    }

    private void TimeToArray()
    {
```

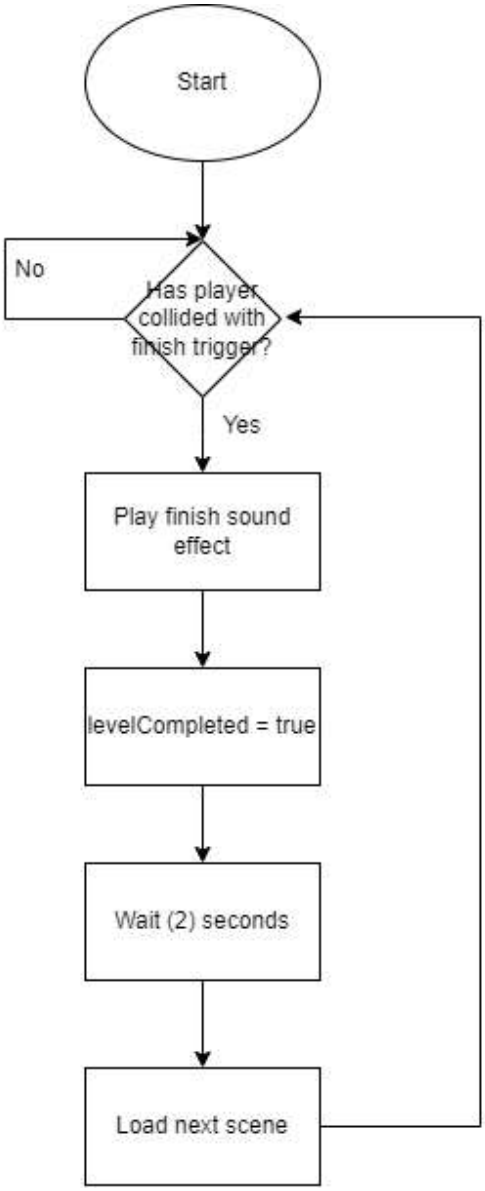
```
        if (!written)
        {
            //Saves local time for level to times array
            times[SceneManager.GetActiveScene().buildIndex - 1] = localTime;
            arrayText.text = string.Join(", ", times);
            written = true;
        }
    }

    private static float[] MergeSort(float[] array)
    {
        //Checks if array is just 1 element long or empty
        if (array.Length <= 1)
        {
            return array;
        }

        //Calculates mid point
        int midPoint = array.Length / 2;

        //Creates left and right half arrays
        float[] left = new float[midPoint];
        float[] right = new float[array.Length - midPoint];

        //Populates left and right arrays with respective contents from array
        Array.Copy(array, 0, left, 0, midPoint);
        Array.Copy(array, midPoint, right, 0, array.Length - midPoint);

        //Recursively sort left and right halves until the array is completely
sorted
        left = MergeSort(left);
        right = MergeSort(right);

        //Sort left and right halves using Merge function and return sorted
array
        return Merge(left, right);
    }

    //Merges two arrays
    private static float[] Merge(float[] left, float[] right)
    {
        float[] result = new float[left.Length + right.Length];

        //Position in left array
```

```
        int i = 0;
        //Position in right array
        int j = 0;
        //Position in "array" array
        int k = 0;

        //Compare elements from the left and right arrays and merge them in
sorted order
        while (i < left.Length && j < right.Length)
        {
            //Comparison
            if (left[i] <= right[j])
            {
                //If left array value is smaller, it is written to "array"
array
                result[k++] = left[i++];
            }
            else
            {
                //If right array value is smaller, it is written to "array"
array
                result[k++] = right[j++];
            }
        }

        //Copy any remaining elements from the left array to "array"
        while (i < left.Length)
        {
            result[k++] = left[i++];
        }

        //Copy any remaining elements from the right array to "array"
        while (j < right.Length)
        {
            result[k++] = right[j++];
        }

        //Returns 1 sorted array
        return result;
    }

}
```

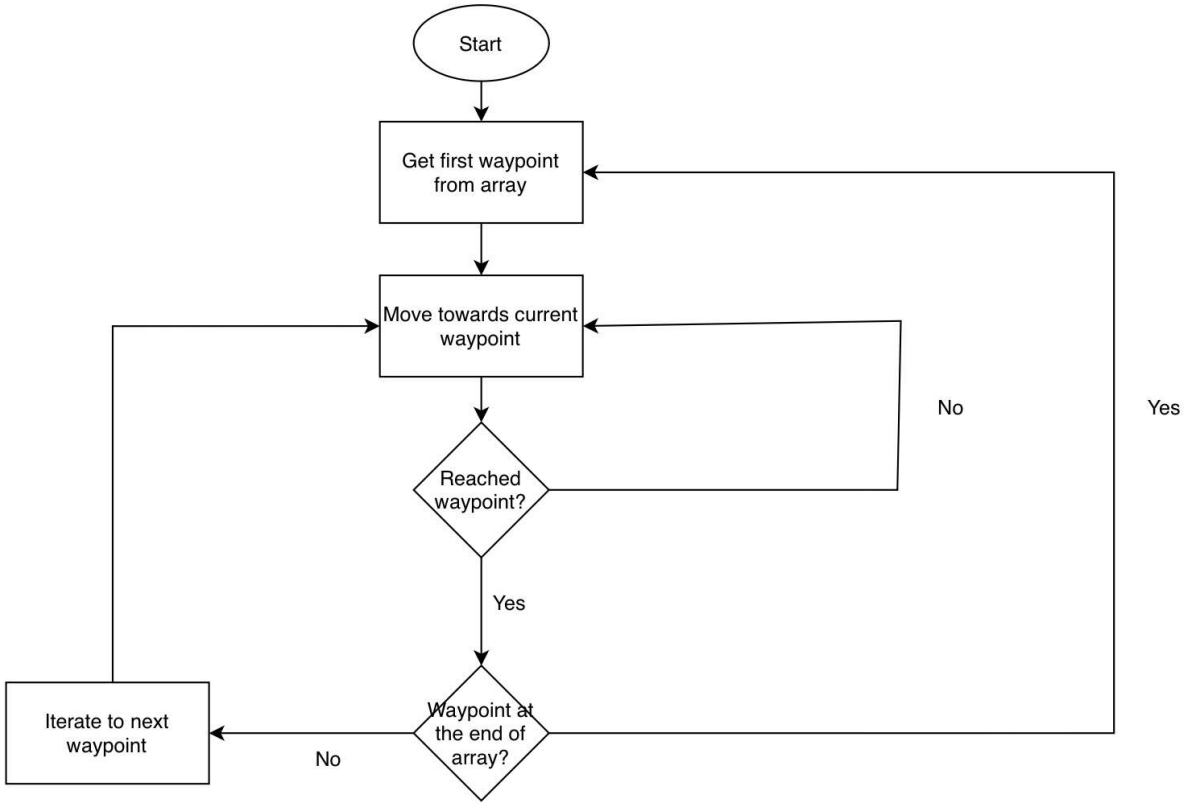
Final (or near final) timer code. This has added:

- A global timer to run across levels (times the total time to complete all levels) as previously mentioned. Runs in addition to a local timer for each level.
- An array of each time for each level, held in array “times”.
- File saving for direct access by the user. Times for each level as well as array - sorted and unsorted - and total time are saved.
- Merge sort to sort the “times” array into order from shortest to longest.
- Base case of the merge sort is when a list has 0 or 1 elements. Time complexity of merge sort is O(n log n) so it is very efficient.

Finish script



Waypoint follower script




```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class WaypointFollower : MonoBehaviour
{
    //Creates a list of waypoints to follow, serialized field so can be changed in the Unity editor
    [SerializeField] private GameObject[] waypoints;
    //Int variable for current waypoint index being moved towards from the list of waypoints
    private int currentWaypointIndex = 0;

    //Float value for speed of moving game object
    [SerializeField] private float speed = 2f;

    //Public because they need to be accessed in the editor
    //Current waypoint game object
```

```
    public GameObject[] Waypoints => waypoints;
    //Current waypoint index (corresponds to list of waypoints)
    public int CurrentWaypointIndex => currentWaypointIndex;

    private void Update()
    {
        //Checks distance between moving game object and waypoint
        if
(Vector2.Distance(waypoints[currentWaypointIndex].transform.position,
transform.position) < 0.1f)
        {
            //Increments currentWaypointIndex by 1 when very close to waypoint
            currentWaypointIndex++;
            //Reverts index to 0 if index exceeds size of list
            if (currentWaypointIndex >= waypoints.Length)
            {
                currentWaypointIndex = 0;
            }
        }
        // Moves the game object each frame
        // Time.deltaTime makes speed move from clock not frame rate
        transform.position = Vector2.MoveTowards(transform.position,
waypoints[currentWaypointIndex].transform.position, Time.deltaTime * speed);
    }
}
```

First waypoint follower script.

Waypoint game objects are added to the dynamic list in the editor.

Disappearing/Recurring platforms

Disappearing platform


```
C/C++
using System.Collections;
```

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

public class DissapearingPlatform : MonoBehaviour
{
    //Allows time to destroy to be changed from Unity engine
    [SerializeField] private float timeToDestroy = 1f;

    //Detects collision
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            //Calls subroutine DisappearAfterDelay()
            StartCoroutine(DisappearAfterDelay());
        }
    }

    private IEnumerator DisappearAfterDelay()
    {
        //Waits for 1 second
        yield return new WaitForSeconds(timeToDestroy);
        //Destroys the platform
        Destroy(gameObject);
    }
}
```

Recurring platform

This is based on the disappearing platform. I started by duplicating the script and changing the class name. Instead of destroying the platform, it disables the platform for timeToRespawn seconds (by deactivating its box collider 2D and dimming the platform colour). The platform then respawns. This will likely replace most, if not all, of the disappearing platforms.

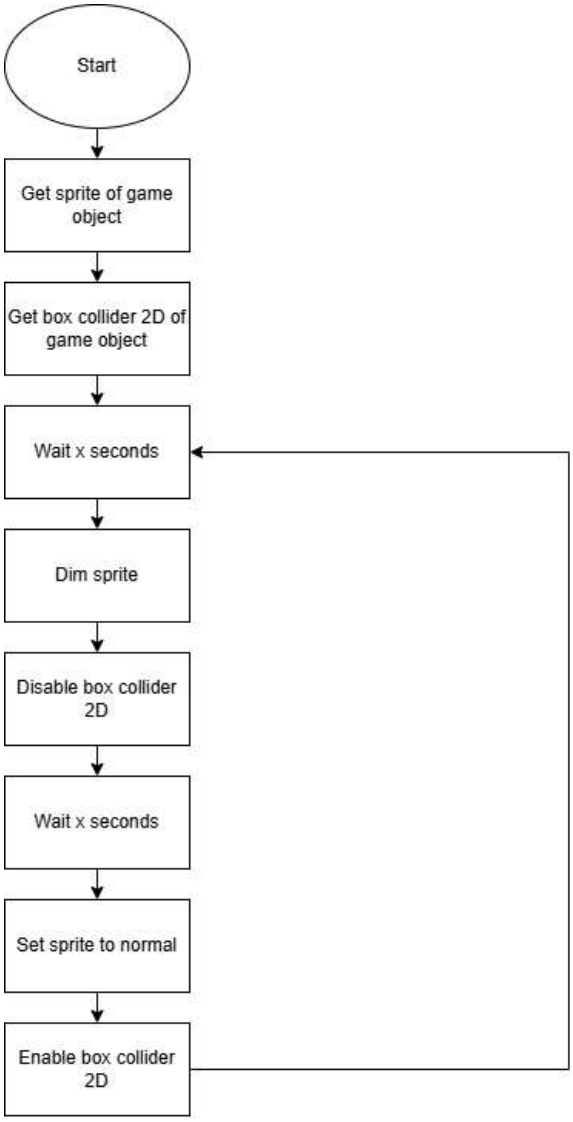
```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RecurringPlatform : MonoBehaviour
{
    //Allows time to destroy to be changed from Unity engine
```

```
    [SerializeField] private float timeToDestroy = 1f;
    [SerializeField] private float timeToRespawn = 1f;

    private SpriteRenderer sprite;
    private BoxCollider2D collider;

    private void Start()
    {
        sprite = GetComponent<SpriteRenderer>();
        collider = GetComponent<BoxCollider2D>();
    }

    //Detects collision
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            //Calls subroutine DisappearAfterDelay()
            StartCoroutine(DisappearAfterDelay());
        }
    }

    private IEnumerator DisappearAfterDelay()
    {
        //Waits for 1 second
        yield return new WaitForSeconds(timeToDestroy);
        //Disables the platform
        //Dims platform sprite
        sprite.color = new Color(100, 100, 100, .5f);
        //Disables platform collider
        collider.enabled = false;
        //Waits for respawn for timeToRespawn seconds
        yield return new WaitForSeconds(timeToRespawn);
        //Reverts colour to normal
        sprite.color = Color.white;
        //Re-enables collider
        collider.enabled = true;
    }
}
```

When active (left), the platform looks the same as the disappearing platform did. However, when temporarily deactivated (right), the platform has its colour dimmed and cannot be stood on as its box collider is disabled.

On/Off Platform script

This script toggles a game object (platform) between active (normal colour and box collider 2D enabled) and inactive (dimmed colour and box collider 2D disabled).



```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OnOffPlatform : MonoBehaviour
{
    //Float to determine how long the game object stays in its current state
before changing
    [SerializeField] private float timePerState = 1f;

    [SerializeField] private float initialDelay = 0f;

    //Creates sprite variable
    private SpriteRenderer sprite;
    //Creates box collier 2D variable
    private BoxCollider2D collider;

    // Start is called before the first frame update
    void Start()
    {
        //Sets sprite and boxcollider 2D variables to the components of the
game object
        sprite = GetComponent<SpriteRenderer>();
        collider = GetComponent<BoxCollider2D>();
        //Start the pltform toggle subroutine
        StartCoroutine(TogglePlatform());
    }


    private IEnumerator TogglePlatform()
    {
        yield return new WaitForSeconds(initialDelay);
        while (true)
        {
            //Waits for time of the active state
            yield return new WaitForSeconds(timePerState);
            //Dims platform colour
            sprite.color = new Color(100, 100, 100, .5f);
            //Disables box collider 2D of game object
            collider.enabled = false;
            //Waits for time of the active state
            yield return new WaitForSeconds(timePerState);
            //Returns platform to normal colour
            sprite.color = Color.white;
```

```
            //Re-enables the box collider 2D
            collider.enabled = true;
        }

    }
}
```

This code allows the platforms to have a set time delay at the start to offset platforms changing state by waiting for initialDelay seconds. initialDelay can be adjusted in the Unity editor, as it is a serialised field.

Platform active (left) and not active (right).

Rotation script

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotate : MonoBehaviour
{
    [SerializeField] private float speed = 2f;
```

```
    // Update is called once per frame
    private void Update()
    {
        transform.Rotate(0, 0, 360 * speed * Time.deltaTime);
    }
}
```

This script rotates a game object (usually a trap/platform) by altering its z transform value at a speed input in the unity editor (2f by default). This has applications for many traps, such as saws and rotating platforms.

Swing

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Swing : MonoBehaviour
{
    //Rate of rotation in degrees per second
    [SerializeField] private float degreesPerSecond = 180f;
    //Wait time in seconds
    [SerializeField] private float waitTime = 1f;
    //Bool to dictate clockwise start
    [SerializeField] private bool clockwise = true;
    //Scale to be applied to rotation to make it start clockwise/anticlockwise
    private float directionScale = 1f;
    //Float of time to wait before swinging starts
    [SerializeField] private float initialDelay = 0f;

    void Start()
    {
        if (clockwise)
        {
            directionScale = 1f;
        }
        else
        {
            directionScale = -1f;
        }
        //Starts rotating on first frame
```

```
        StartCoroutine(Rotate());
    }

    IEnumerator Rotate()
    {
        yield return new WaitForSeconds(initialDelay);
        while (true)
        {
            //Rotate 180 degrees clockwise in z axis
            yield return StartCoroutine(RotateByAngle(Vector3.forward, 180f *
directionScale));


            //Waits for specified time
            yield return new WaitForSeconds(waitTime);

            //Rotate 180 degrees anticlockwise in z axis
            yield return StartCoroutine(RotateByAngle(Vector3.forward, -180f *
directionScale));

            //Waits for specified time
            yield return new WaitForSeconds(waitTime);
        }
    }

    IEnumerator RotateByAngle(Vector3 axis, float angle)
    {
        //Rotates the game object
        float targetAngle = Mathf.Abs(angle);
        float rotatedAngle = 0f;

        while (rotatedAngle < targetAngle)
        {
            float step = degreesPerSecond * Time.deltaTime;
            if (rotatedAngle + step > targetAngle)
            {
                //Avoids swing over-rotating
                step = targetAngle - rotatedAngle;
            }
            transform.Rotate(axis, Mathf.Sign(angle) * step);
            rotatedAngle += step;
            yield return null;
        }
    }
```

```
}
```

This script rotates a game object (will be a swing for the player to move on) by 180 degrees at a time at a speed decided in the Unity editor (180f degrees per second by default). Initial time delay (to offset the stage of swinging of different platforms) and the time the swing is stationary after a swing can also be changed in the Unity editor. You can also change the swing to start clockwise or anticlockwise in the Unity editor.

Flying item

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyingItem : MonoBehaviour
{
    //Gets animator
    private Animator anim;
    //Gets Rigidbody2D
    private Rigidbody2D rb;
    //X and Y direction values
    private float dirX = 0f;
    private float dirY = 0f;
    //Movement speed can be changed in Unity editor
    [SerializeField] private float moveSpeed = 5f;
    //Boolean for whether flying item is active
    private bool active;

    private void Start()
    {
        //Gets components on start frame
        anim = GetComponent<Animator>();
        anim.SetTrigger("Idle");
        rb = GetComponent<Rigidbody2D>();
        dirX = Input.GetAxisRaw("Horizontal");
        dirY = Input.GetAxisRaw("Vertical");
        rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
    }

    private void OnTriggerStay2D(Collider2D collision)
    {
```

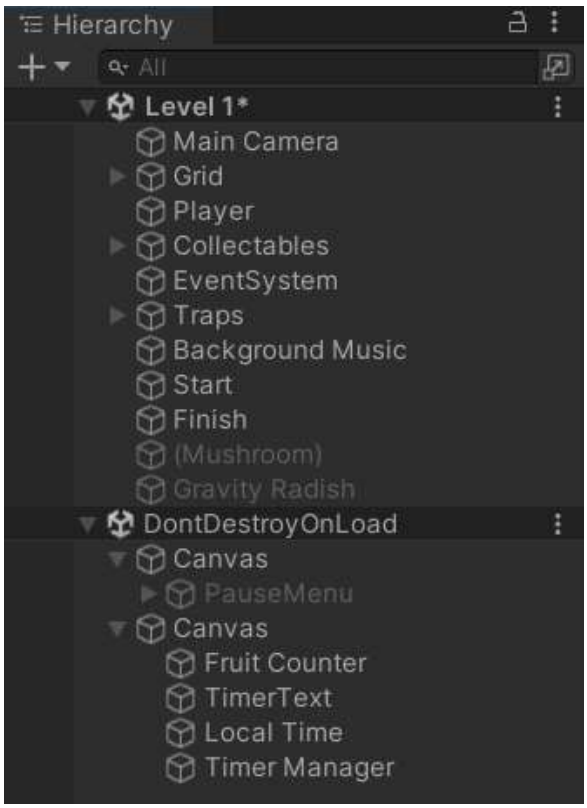
```
        //On collision activates animation
        anim.SetTrigger("Active");
        //Sets active boolean to true
        active = true;
    }

    private void OnTriggerExit2D(Collider2D collision)
    {
        //On end of collision deactivates animation *
        anim.SetTrigger("Idle");
        //Sets active boolean to false
        active = false;
    }

    private void Update()
    {
        //Calls movement subroutine every frame if flying item is active
        if (active)
        {
            Movement();
        }
    }

    //Allows X and Y movement of flying item
    //Not affected by gravity so can move freely
    private void Movement()
    {
        //Takes X and Y input
        dirX = Input.GetAxisRaw("Horizontal");
        dirY = Input.GetAxisRaw("Vertical");
        //Apply input to vector, proportional to moveSpeed
        rb.velocity = new Vector2(dirX * moveSpeed, dirY * moveSpeed);
    }

}
```

This code implements a “flying item” similar to the cloud from Super Mario World. It allows the player to control the item and fly through the air. When the player enters the trigger zone of the flying item the player now controls the flying item, and the player itself is contained within the box colliders of the flying item so will not fall out.

Persistent UI

A script to make sure UI elements e.g. timers, fruit counter and pause menu are carried over between levels.

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PersistentUI : MonoBehaviour
{
    private void Awake()
    {
        GameObject[] objs = GameObject.FindGameObjectsWithTag("PersistentUI");

        if (objs.Length > 2)
        {
            Destroy(gameObject);
        }

        DontDestroyOnLoad(gameObject);
    }
}
```



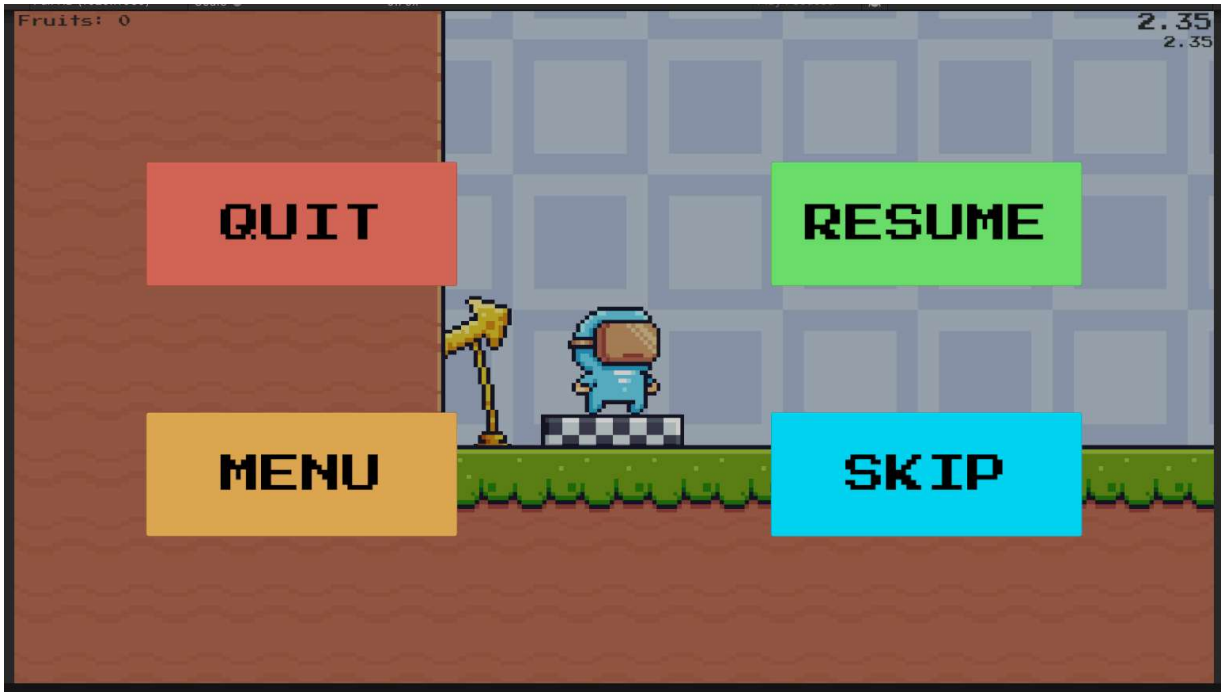
In the hierarchy during gameplay, the UI canvases for counters/timers and the pause menu are located separately in the DontDestroyOnLoad section. Everything else is only present in level 1.

Pause Menu

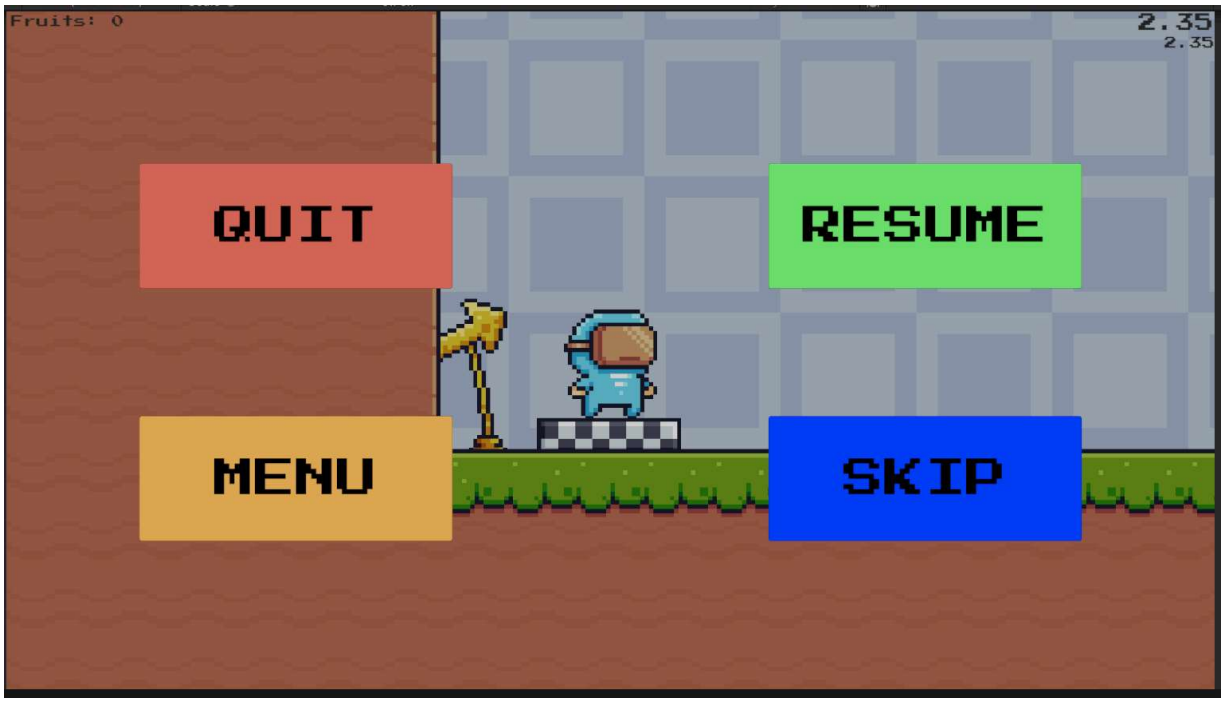
```C/C++
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    public static bool gameIsPaused = false;
    public GameObject pauseMenuUI;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (gameIsPaused)
            {
                Resume();
            }

            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        Time.timeScale = 1f;
        gameIsPaused = false;
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        gameIsPaused = true;
```

```
    }

    public void LoadMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }

    public void QuitGame()
    {
        Application.Quit();
    }

    public void SkipLevel()
    {
        //Loads the level of the next build index
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        Resume();
    }
}
```

Pause Menu algorithm after later test. Each button has a corresponding function in this script, which is made public so it can be attached to the button. Escape button can be used to pause/unpause the game.



Pause menu post test and solution.



Colour of button changes when it is hovered over.

Recall Stack

This feature allows a player to set a spawn point as long as they are on the ground, and then they can teleport to the last recall point they set. X and Y coordinates are held in their

corresponding stacks, and when recall is triggered, the last recall point is popped from the stack. A sound plays when a recall point is set and when the player teleports to the last recall point.
Pseudocode:

```
Unset
stackx = []
stacky = []
if set:
       stackx.push(current x coordinate)
       stacky.push(current y coordinate)
if pop:
       playerPosition = (stackx.pop, stacky.pop)
```

First draft code:

```
C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Recall : MonoBehaviour
{

    private Stack<float> stackX = new Stack<float>();
    private Stack<float> stackY = new Stack<float>();

    private BoxCollider2D coll;
    [SerializeField] private LayerMask jumpableGround;

    [SerializeField] private Rigidbody2D rb;

    [SerializeField] private AudioSource spawnSet;
    [SerializeField] private AudioSource recall;

    private void Start()
    {
        coll = GetComponent<BoxCollider2D>();
    }

    void Update()
    {
```

```
        if (Input.GetButtonDown("Fire1") && IsGrounded())
        {
            spawnSet.Play();
            stackX.Push(transform.position.x);
            stackY.Push(transform.position.y);
            Debug.Log("Recall set");
        }

        if (Input.GetButtonDown("Fire2") && stackX.Count != 0 && stackY.Count
!= 0)
        {
            recall.Play();
            rb.MovePosition(new Vector3(stackX.Pop(), stackY.Pop(),
transform.position.z));
        }
    }

    private bool IsGrounded()
    {
        //Box colliders
        //Creates box cast around player collider box to check if player is on
the ground/ near walls
        //Used so player can only jump on ground, not above ground or by walls
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f,
Vector2.down, 0.1f, jumpableGround);
    }
}
```

Quote API Import

Code imports an API containing an inspirational quote and displays it on the end screen of the game. Uses the zen quotes API. First draft.

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Allows JSON parsing
using UnityEngine.Networking;
//Allows text to be changed
using UnityEngine.UI;
```

```
public class ImportAPI : MonoBehaviour
{
    //Text for quote and author to be written to added in Unity editor
    [SerializeField] private Text quoteText;
    //API URL to be used
    private const string API_URL = "https://zenquotes.io/api/random";

    void Start()
    {
        StartCoroutine(FetchQuote());
    }

    private IEnumerator FetchQuote()
    {
        using (UnityWebRequest webRequest = UnityWebRequest.Get(API_URL))
        {
            //Requests quote
            yield return webRequest.SendWebRequest();
            //JSON parsing
            //Gets response
            string jsonResponse = webRequest.downloadHandler.text;
            //Gets quote part
            string quote = jsonResponse.Split(new[] { "\"q\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
            //Gets author part
            string author = jsonResponse.Split(new[] { "\"a\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
            //Concatenates author and quote with space and outputs to the
quoteText
                quoteText.text = quote + author;
            }
        }
    }
}
```

Array of default quotes added in case API request fails. Quote will be randomly selected. Formatting of quote and author fixed by adding a gap between them when concatenating the output string.

```
C/C++
using System.Collections;
using System.Collections.Generic;
```

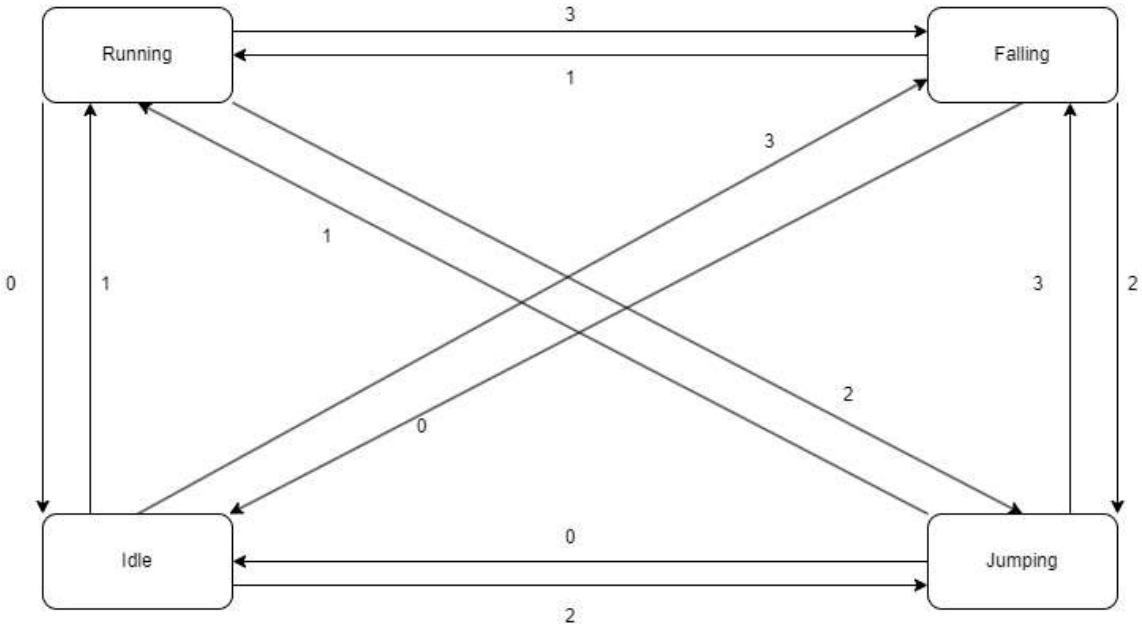
```
using UnityEngine;
//Allows JSON parsing
using UnityEngine.Networking;
//Allows text to be changed
using UnityEngine.UI;

public class ImportAPI : MonoBehaviour
{
    //Text for quote and author to be written to added in Unity editor
    [SerializeField] private Text quoteText;
    //API URL to be used
    private const string API_URL = "https://zenquotes.io/api/random";
    //Array of default quotes to be used if API request fails
    private string[] defaultQuotes = {"Have a great day!", "Well played!",
"Play again soon!"};

    void Start()
    {
        StartCoroutine(FetchQuote());
    }

    private IEnumerator FetchQuote()
    {
        using (UnityWebRequest webRequest = UnityWebRequest.Get(API_URL))
        {
            //Requests quote
            yield return webRequest.SendWebRequest();

            //If API request fails, a random quote from defaultQuotes array is
used instead
            if (webRequest.result == UnityWebRequest.Result.ConnectionError ||
webRequest.result == UnityWebRequest.Result.ProtocolError)
            {
                quoteText.text = defaultQuotes[Random.Range(0,
defaultQuotes.Length)];
            }
            else
            {
                //JSON parsing

                //Gets response
                string jsonResponse = webRequest.downloadHandler.text;
                //Gets quote part
```

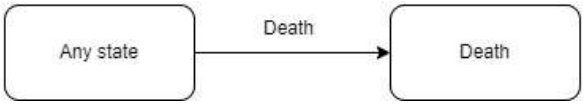
```
                string quote = jsonResponse.Split(new[] { "\"q\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
                //Gets author part
                string author = jsonResponse.Split(new[] { "\"a\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
                //Concatenates author and quote with space and outputs to the
quoteText
                quoteText.text = quote + " - " + author;
            }
        }
    }
}
```

Animations

Most traps and platforms have constant animations, but some change.

Player





0 = No velocity
1 = Horizontal velocity and y velocity not negative
2 = Increasing y velocity and not on ground/platform
3 = Decreasing y velocity

```
C/C++
private enum MovementState {Idle, Running, Jumping, Falling}
```

Numbers correspond to the MovementState array in the player movement script. Animation state is changed in player movement script based on x and y velocity. Any state can go to death when triggered from player life script.

Idle

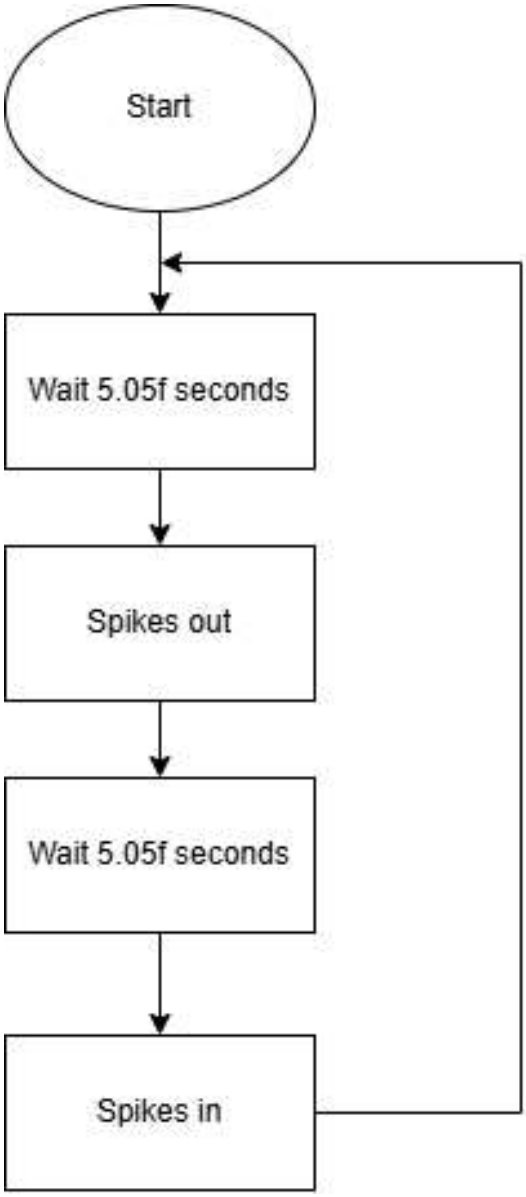
Running

Jump

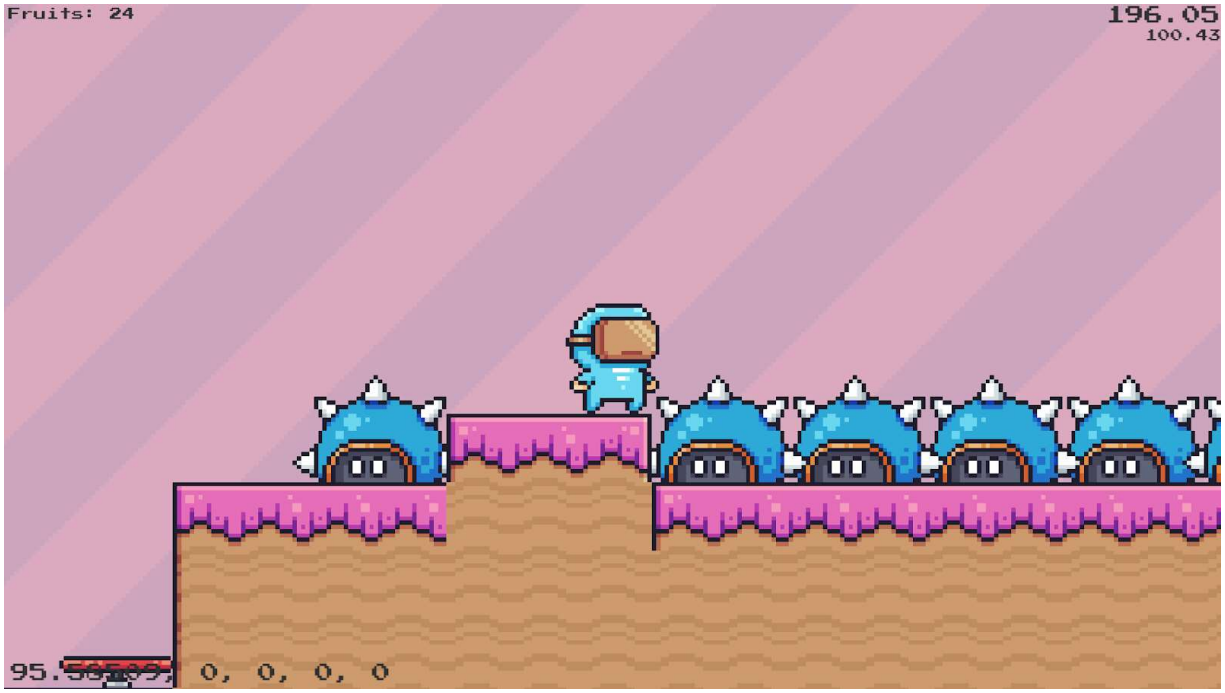
Fall

Spike Turtle

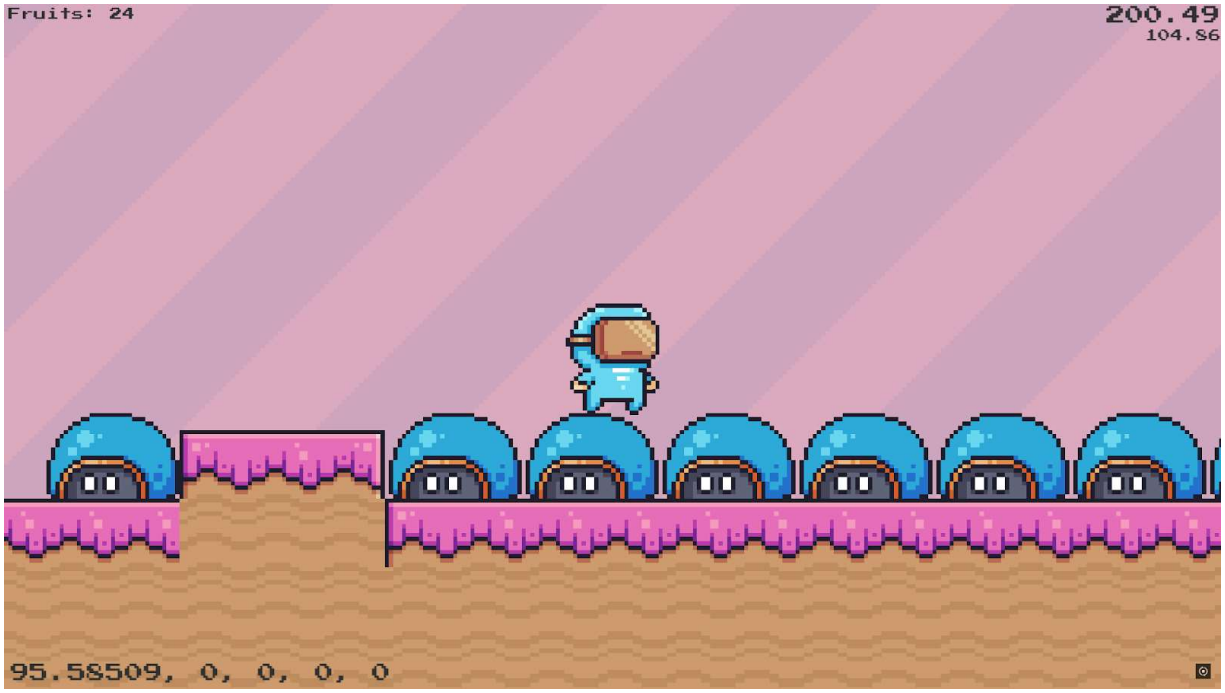
Spike turtle cycles between being a trap with spikes out or normal ground without having its spikes out, every 5.05f seconds.


Start
Wait 5.05f seconds
Spikes out
Wait 5.05f seconds
Spikes in

Flow chart.



Spikes out.

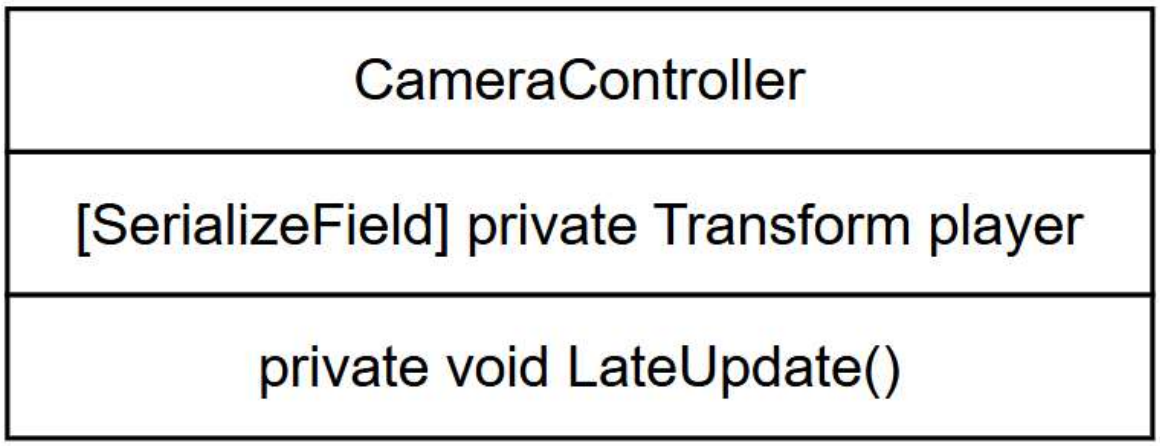


Spikes in.

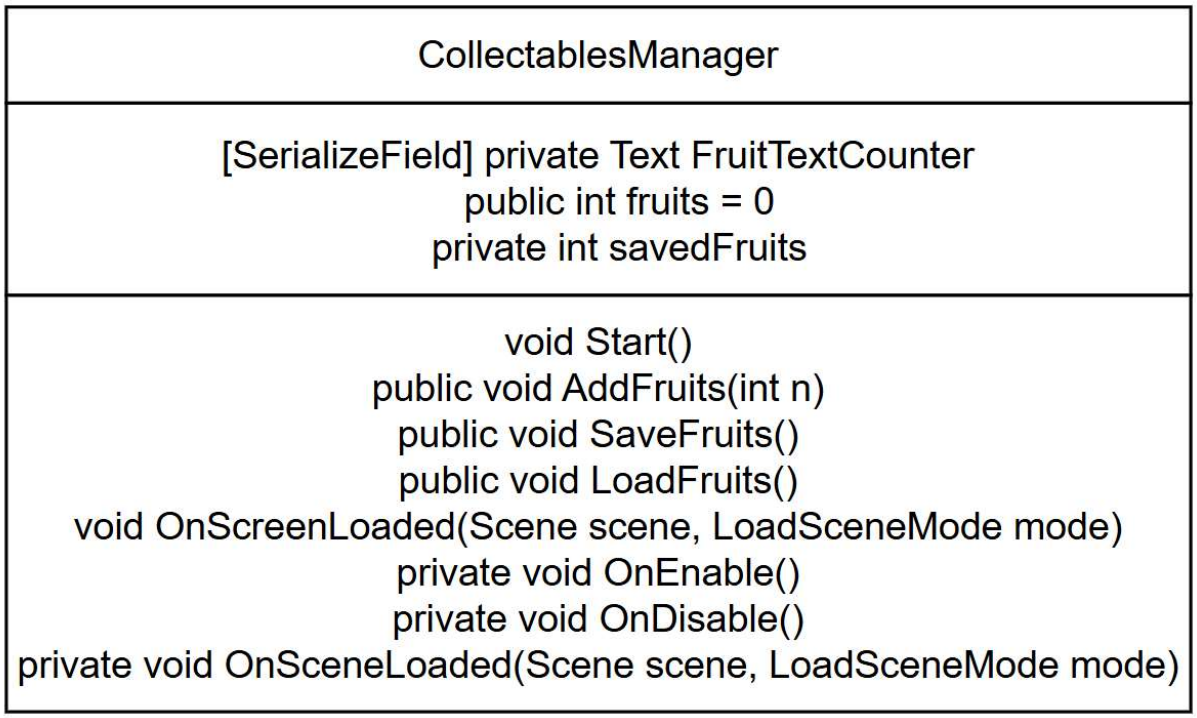
Class Diagrams

These class diagrams show the class name, variables and subroutines for each class in the game.

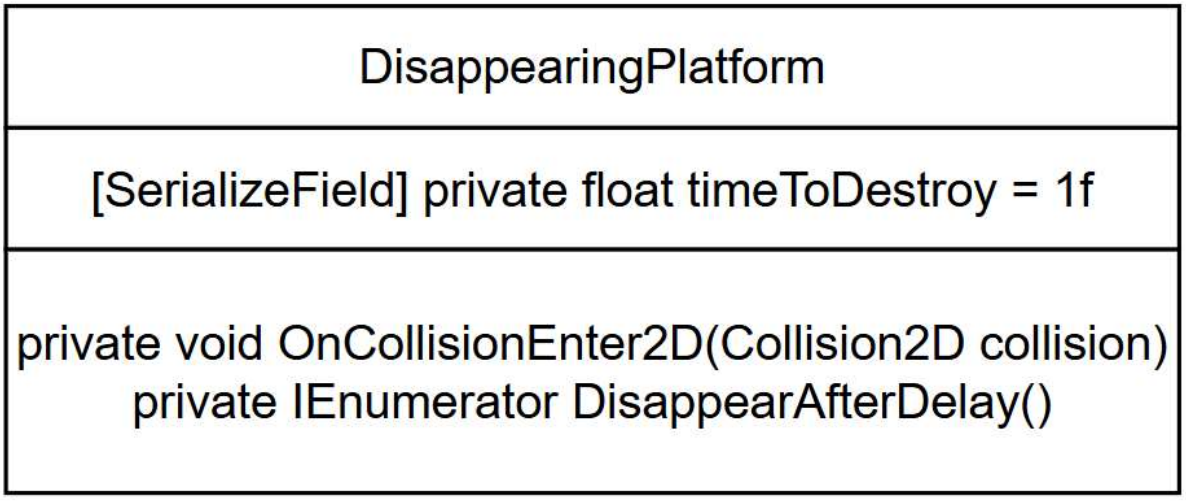
Camera Controller



Collectables Manager

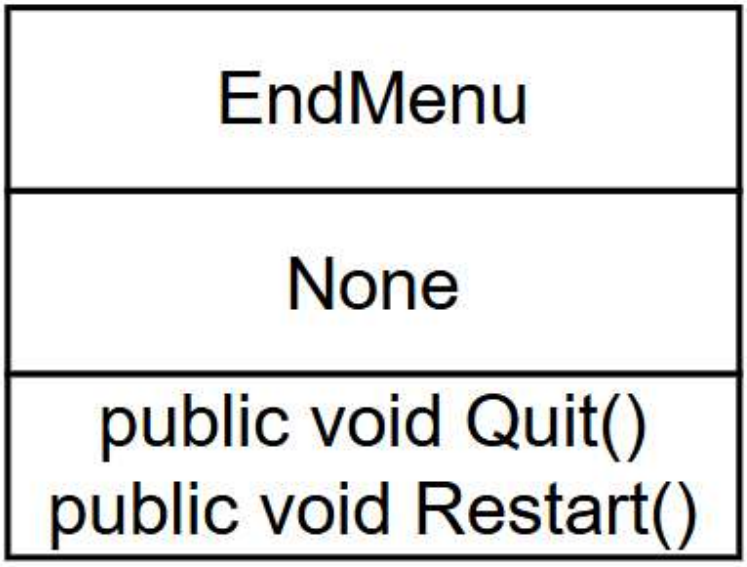


Disappearing Platform*



*Later replaced by RecurringPlatform class.

End Menu



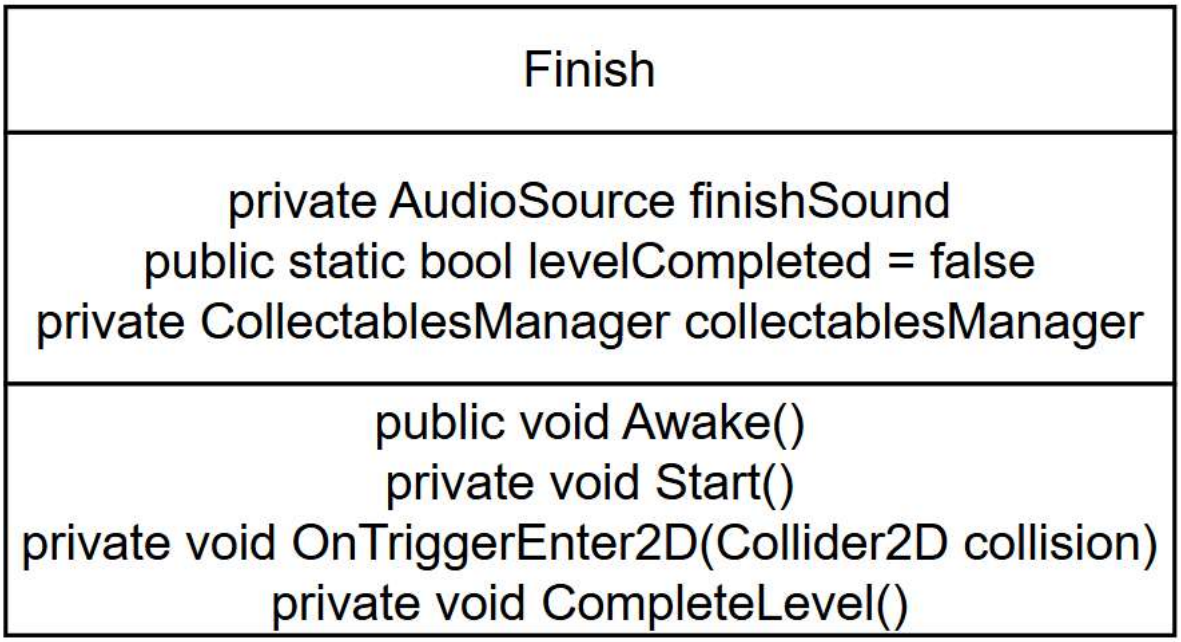
Enemy Death

EnemyDeath
private Animator anim [SerializeField] private float animationTime = 1f
void Start() private void OnTriggerEnter2D(Collider2D collision) private IEnumerator DisappearAfterDelay()

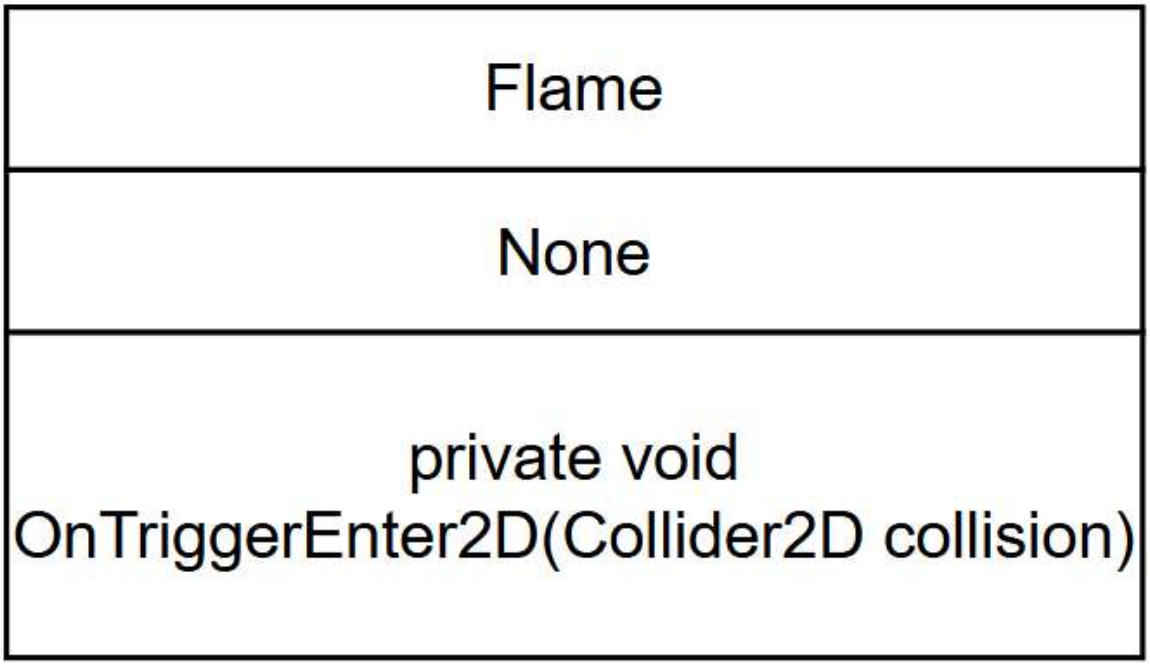
Falling Spike Head

FallingSpikeHead
private Rigidbody2D rb [SerializeField] private float dropForce = -10f
private void Start() private void OnTriggerEnter2D(Collider2D collision)

Finish



Flame



Floating Enemy

FloatingEnemy
[SerializeField] private float timeToMove = 1f [SerializeField] private float swayMagnitude = 3f private Rigidbody2D rb private bool movingUp = true;
void Start() private IEnumerator Sway()

Flying Item

FlyingItem
private Animator anim private Rigidbody2D rb private float dirX = 0f private float dirY = 0f [SerializeField] private float moveSpeed = 5f private bool active
private void Start() private void OnTriggerStay2D(Collider2D collision) private void OnTriggerExit2D(Collider2D collision) private void Update() private void Movement()

Import API

ImportAPI
[SerializeField] private Text quoteText private const string API_URL = "https://zenquotes.io/api/random" private string[] defaultQuotes = {"Have a great day!", "Well played!", "Play again soon!"}
void Start() private IEnumerator FetchQuote()

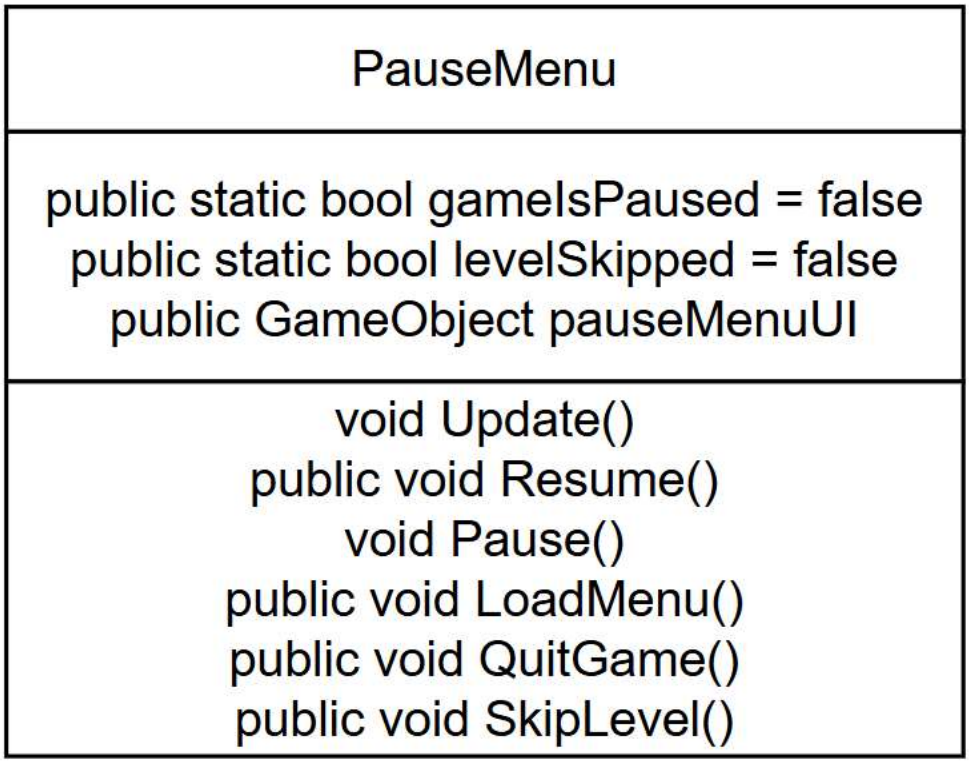
Item Collector

ItemCollector
private int fruits = 0 public Text FruitsText [SerializeField] private AudioSource collectionSoundEffect private CollectablesManager collectablesManager
public void Awake() private void OnTriggerEnter2D(Collider2D collision)

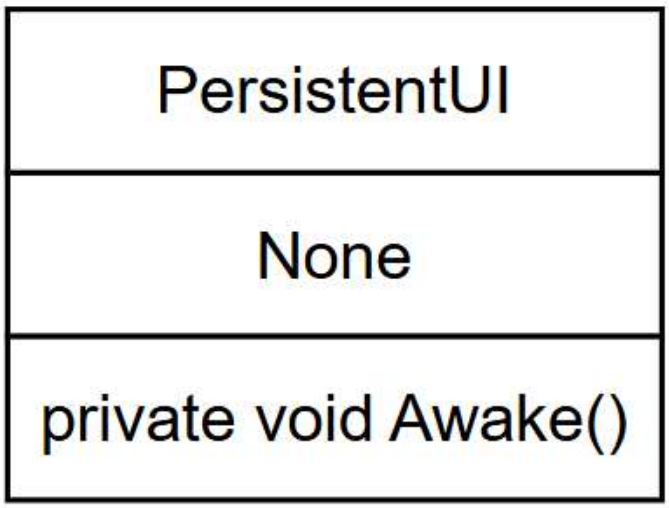
On/Off Platform

OnOffPlatform
[SerializeField] private float timePerState = 1f [SerializeField] private float initialDelay = 0f private SpriteRenderer sprite private BoxCollider2D collider
void Start() private IEnumerator TogglePlatform()

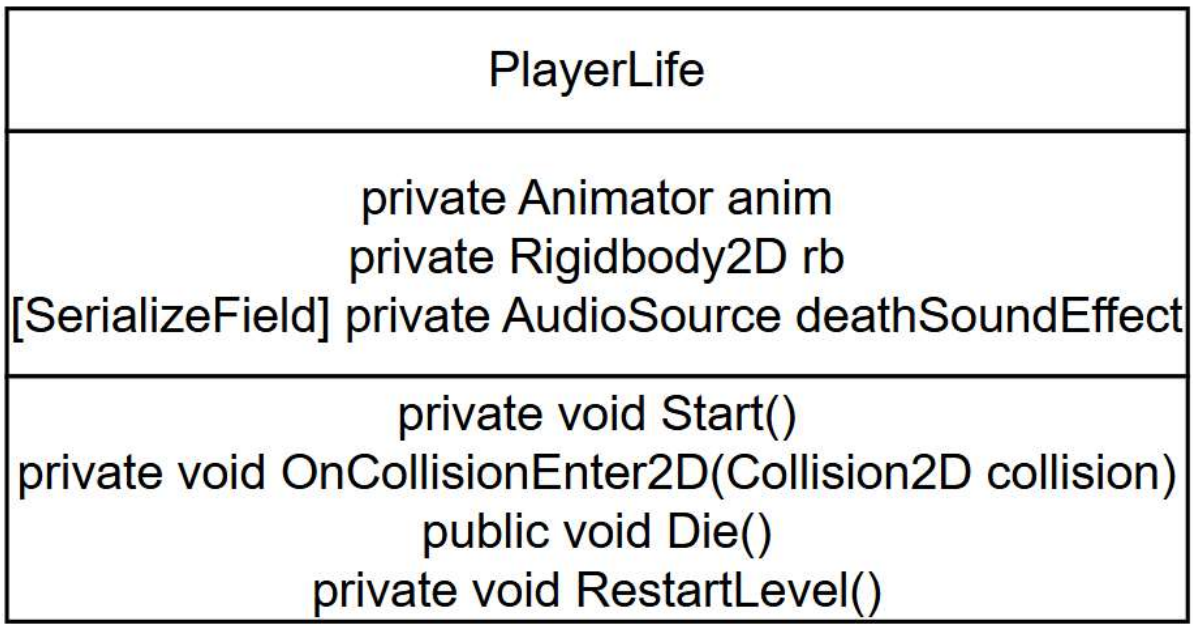
Pause Menu



Persistent UI



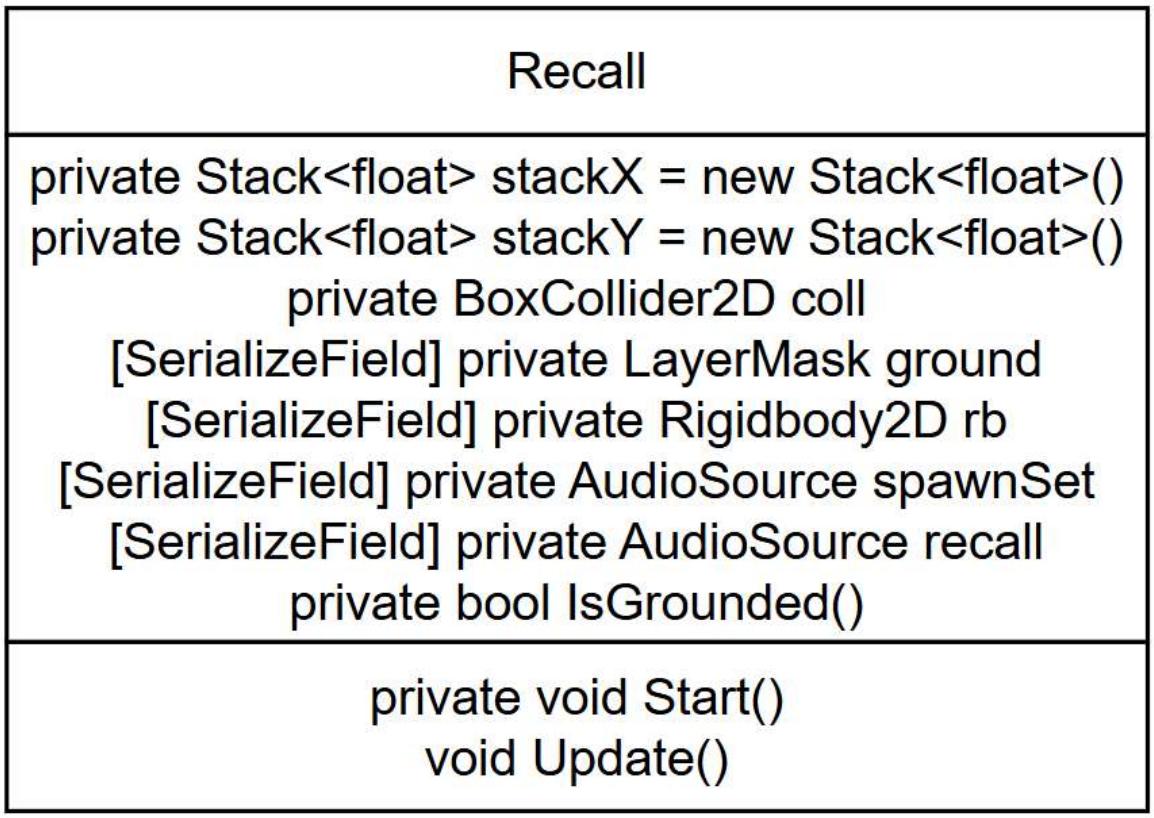
Player Life


PlayerLife
private Animator anim
private Rigidbody2D rb
[SerializeField] private AudioSource deathSoundEffect
private void Start()
private void OnCollisionEnter2D(Collision2D collision)
public void Die()
private void RestartLevel()

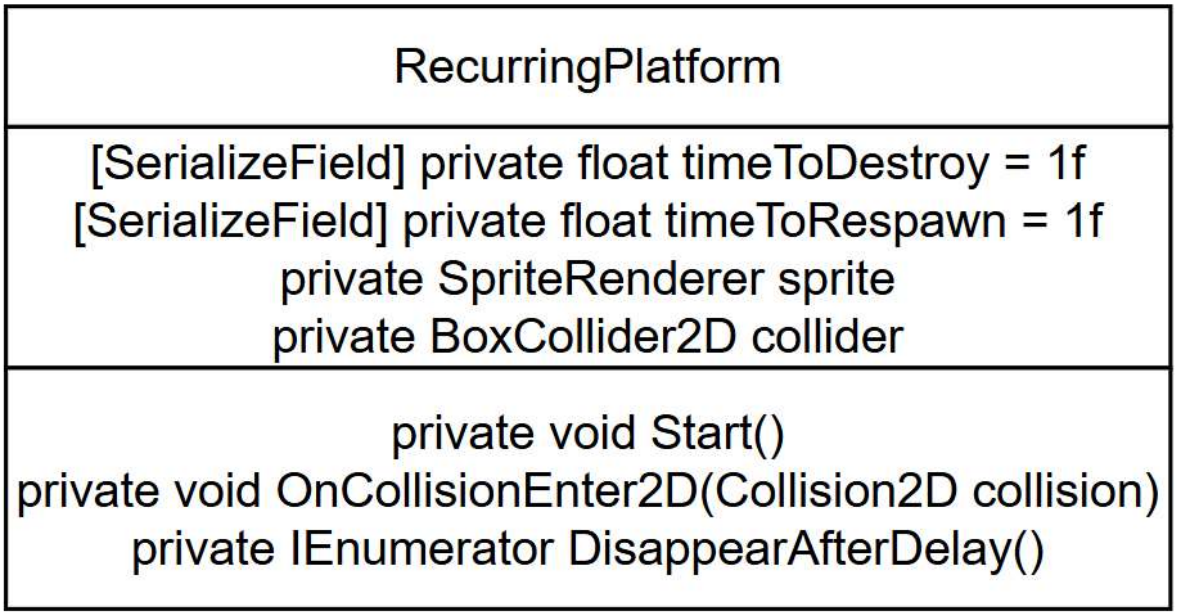
Player Movement

PlayerMovement
private Rigidbody2D rb private Animator anim private float dirX = 0f private SpriteRenderer sprite [SerializeField] public float playerMoveSpeed = 7f [SerializeField] private float jumpForce = 14f private enum MovementState {Idle, Running, Jumping, Falling} private BoxCollider2D coll [SerializeField] private LayerMask jumpableGround [SerializeField] private AudioSource jumpSoundEffect private bool IsGrounded()
private void Start() private void Update() private void UpdateAnimationState()

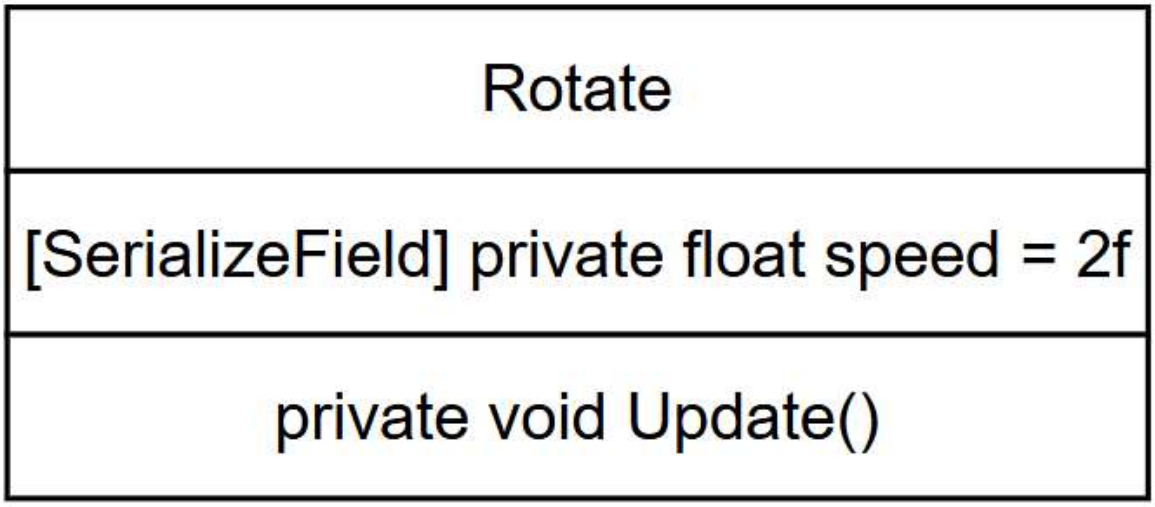
Recall


Recall
private Stack<float> stackX = new Stack<float>()
private Stack<float> stackY = new Stack<float>()
private BoxCollider2D coll
[SerializeField] private LayerMask ground
[SerializeField] private Rigidbody2D rb
[SerializeField] private AudioSource spawnSet
[SerializeField] private AudioSource recall
private bool IsGrounded()
private void Start()
void Update()

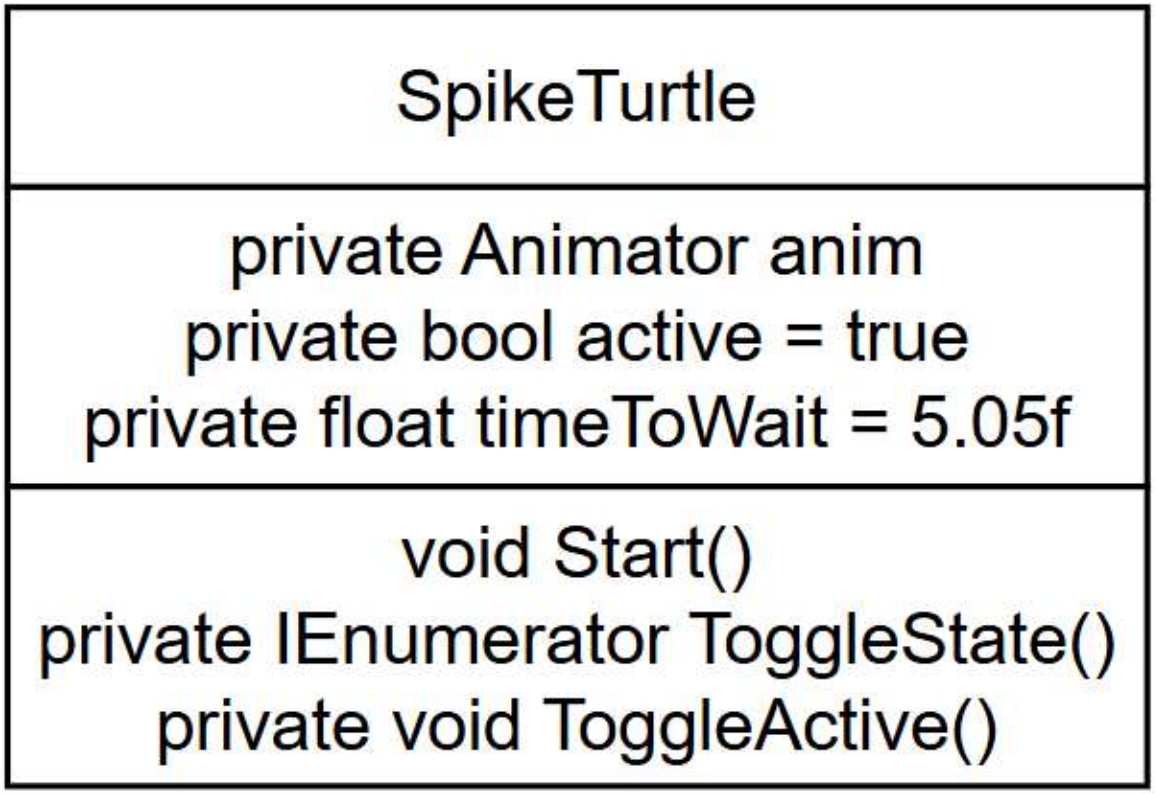
Recurring Platform



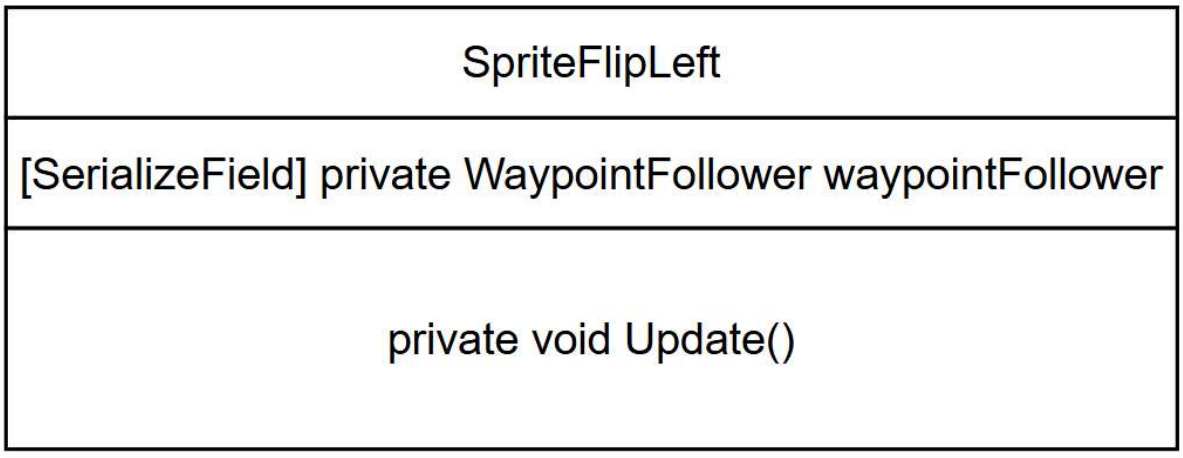
Rotate



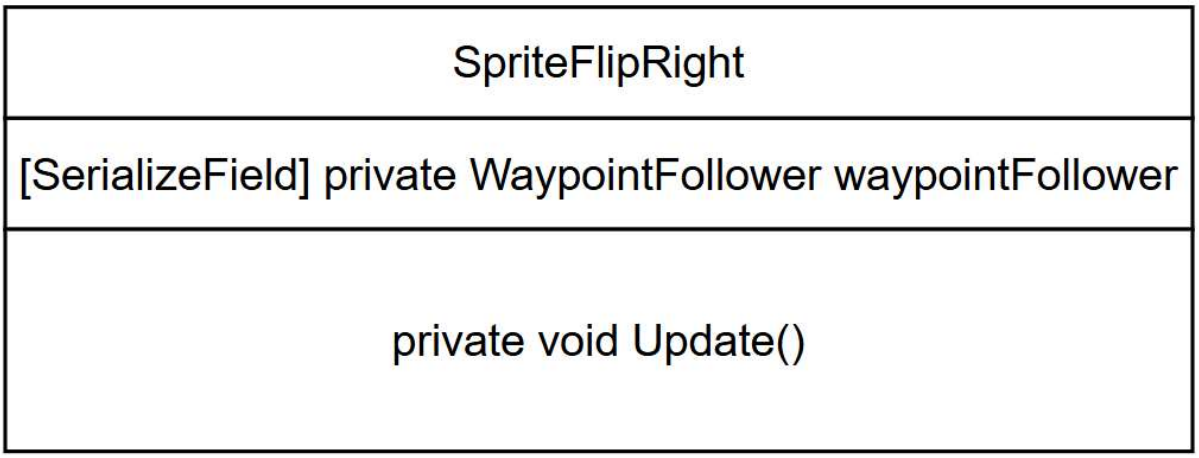
Spike Turtle



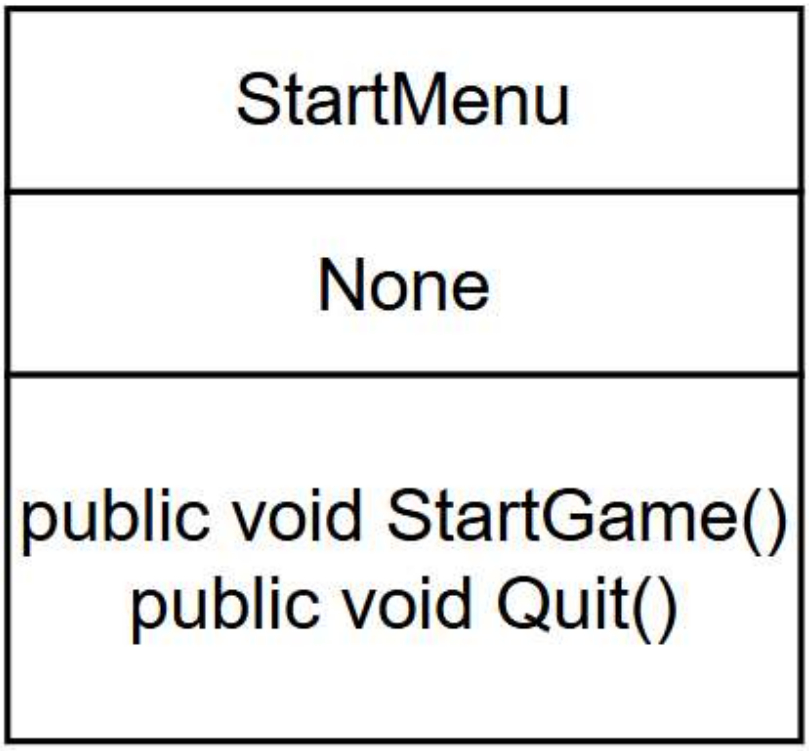
Sprite Flip Left



Sprite Flip Right



Start Menu



Sticky Platform

StickyPlatform
None
private void OnTriggerEnter2D(Collider2D collision) private void OnTriggerExit2D(Collider2D collision)

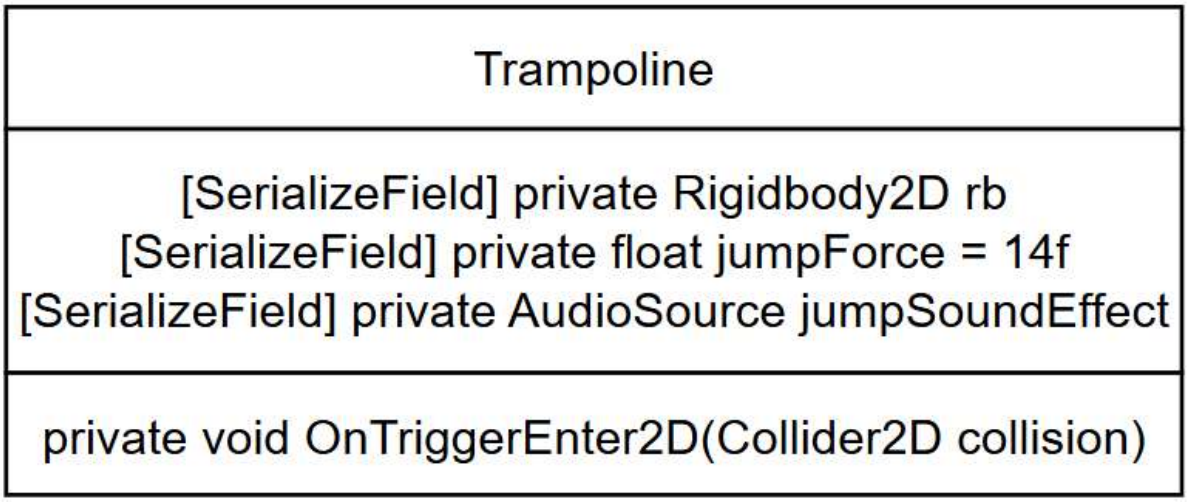
Swing

Swing
[SerializeField] private float degreesPerSecond = 180f [SerializeField] private float waitTime = 1f [SerializeField] private bool clockwise = true private float directionScale = 1f [SerializeField] private float initialDelay = 0f
void Start() IEnumerator Rotate() IEnumerator RotateByAngle(Vector3 axis, float angle)

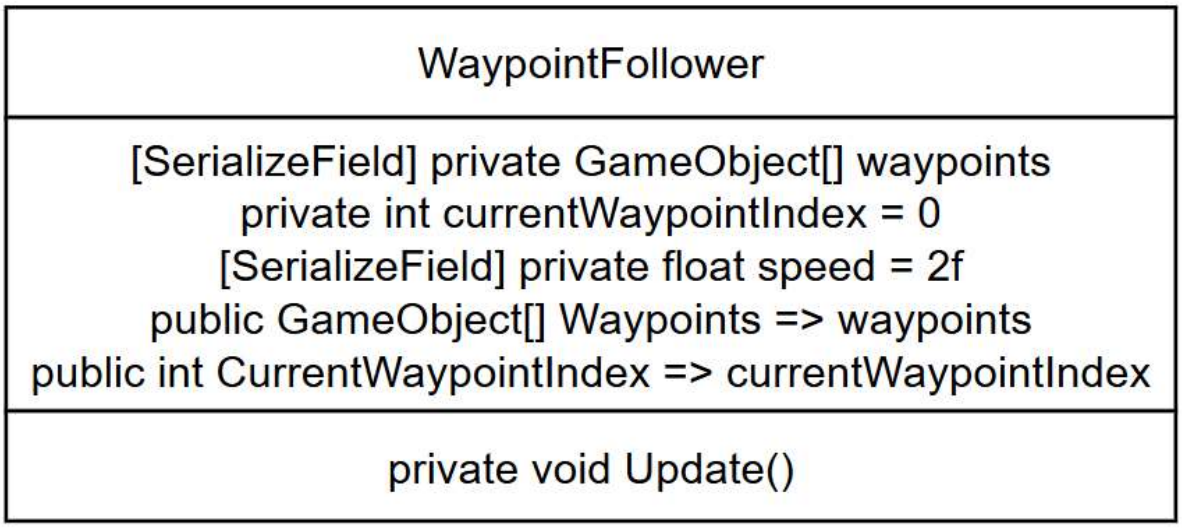
Timer

Timer
public Text timerText [SerializeField] private Text localTimeText [SerializeField] private Text arrayText private float currentTime = 0f private float localTime = 0f private bool TimeEnd = false private bool skipped = false public Finish finishScript public PauseMenu pauseMenu private float[] times = new float[5] private float [] sortedTimes = new float[5] private bool written = false private bool fileWritten = false
private void Update() private void SaveTimesToFile() private void IncrementTime() private void TimeToArray() private static float[] MergeSort(float[] array) private static float[] Merge(float[] left, float[] right) private void OnEnable() private void OnDisable() private void OnSceneLoaded(Scene scene, LoadSceneMode mode) private void ResetTimesArray()

Trampoline



Waypoint Follower



Layers

I have utilised the following layers:

- Default
- Ground
- UI

Anything in the ground layer can be jumped on. This layer allows a game object to check if it is on the ground with the following function:

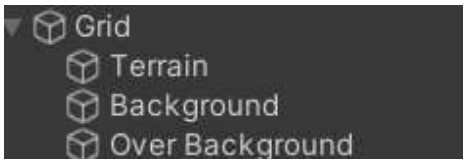
```C/C++
private bool IsGrounded()
    {
        //Box colliders
        //Creates box cast around player collider box to check if player is on the ground/ near walls
        //Used so player can only jump on ground, not above ground or by walls
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f, Vector2.down, 0.1f, jumpableGround);
    }
```

IsGrounded() function

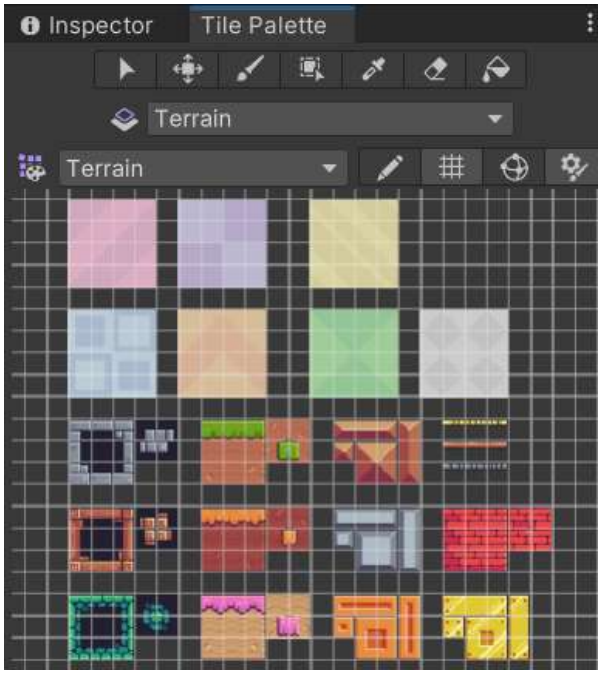
The UI layer displays above all other layers so the UI is never hidden behind other game objects. Anything else is in the default layer.

Grid

The grid is used to place tiles from the tile map into the game. Under the grid, there is terrain, background and over background. Terrain is the front grid, over background is in the middle grid and background is the back grid. This ensures terrain is always at the front and the over background is always in front of the background.



Grid in hierarchy

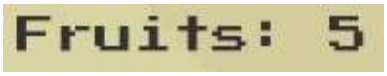


Tile palette

Terrain always visible, over background (sticks which saws move on) in front of background

Collectables

Each time a fruit is touched by the player, it is destroyed and increases the fruit counter by 1, each level has its own unique type of fruit for the player to collect.



Fruit counter.

Cherry, level 1.

Watermelon, level 2.

Orange, level 3.

Kiwi, level 4.

Apple, level 5.

Trap/Platform design

All traps are tagged as “Traps”. The player dies when it collides with anything tagged as “Traps”.

Ground enemies

Ground enemies such as the angry pig have the following components:

- Sprite renderer
- Animator
- Waypoint follower
- Sprite flip left
- Box collider 2D

Angry pig with waypoints (marked with pink diamonds)

Happy pig with waypoints (marked with pink diamonds)

Flying enemies

Flying enemies such as the bat have the following components:

- Sprite renderer
- Animator
- Rigidbody 2D
- Box collider 2D
- Waypoint follower
- Sprite flip left*
- Floating enemy*

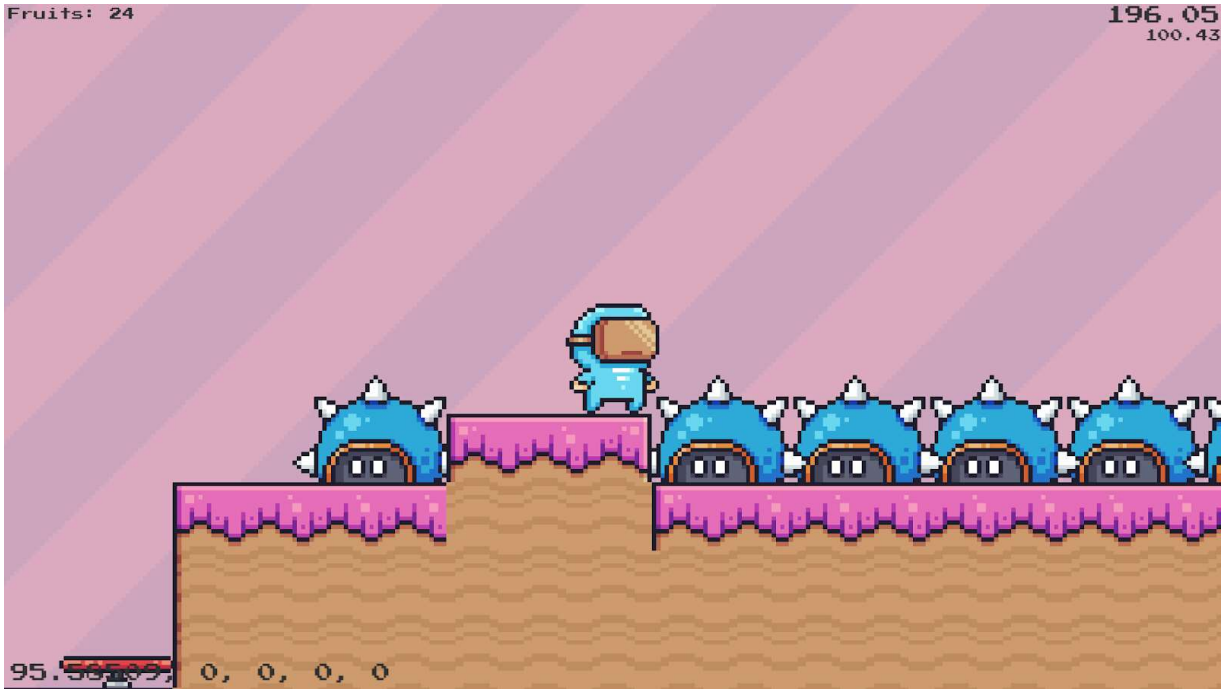
*Not present on the bee because it oscillates vertically

Bat with its waypoints

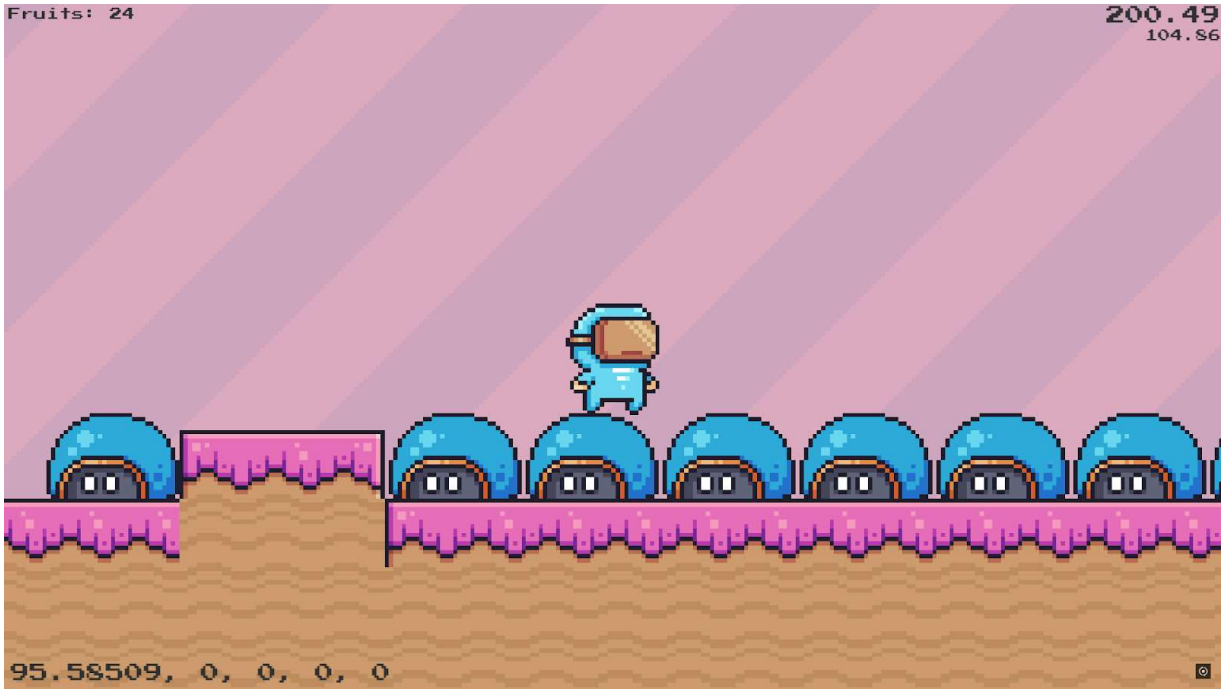
Spike Turtle

The spike turtle has the following components:

- Sprite renderer
- Animator
- Circle collider 2D
- Spike turtle



Spike turtles active

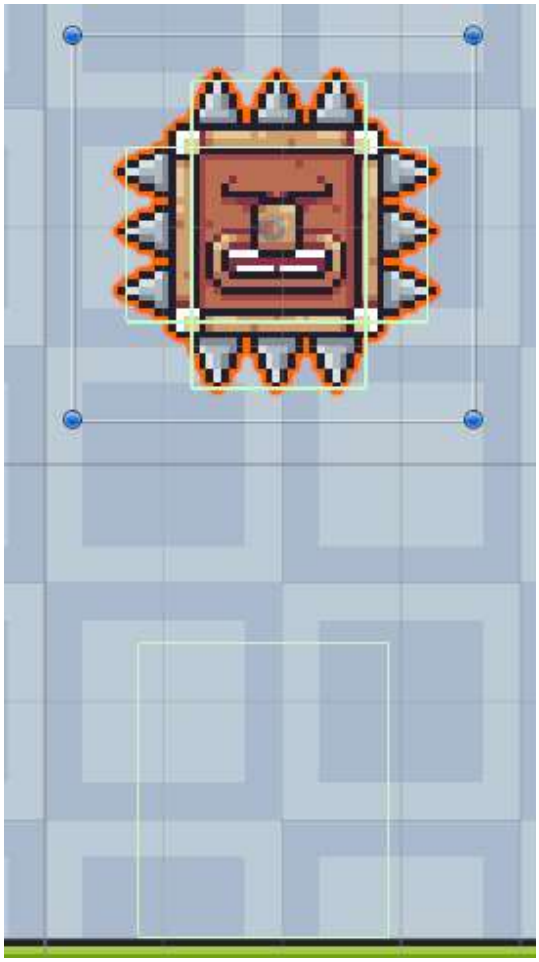


Spike turtles not active so can be walked on and jumped on

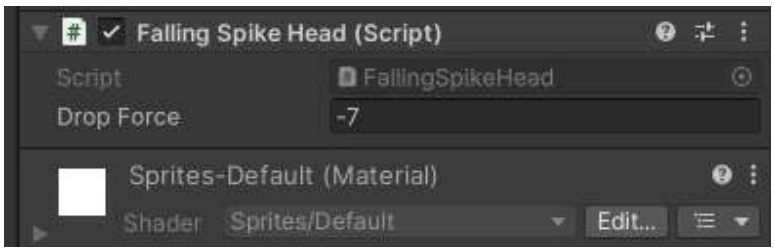
Falling Spike Head

The falling spike head falls when the player walks in its trigger box. It has the following components:

- Sprite renderer
- Animator
- Rigidbody2D
- Box collider 2D
    - 2 for collision
    - 1 for trigger box
- Falling spike head



Falling spike head and trigger box

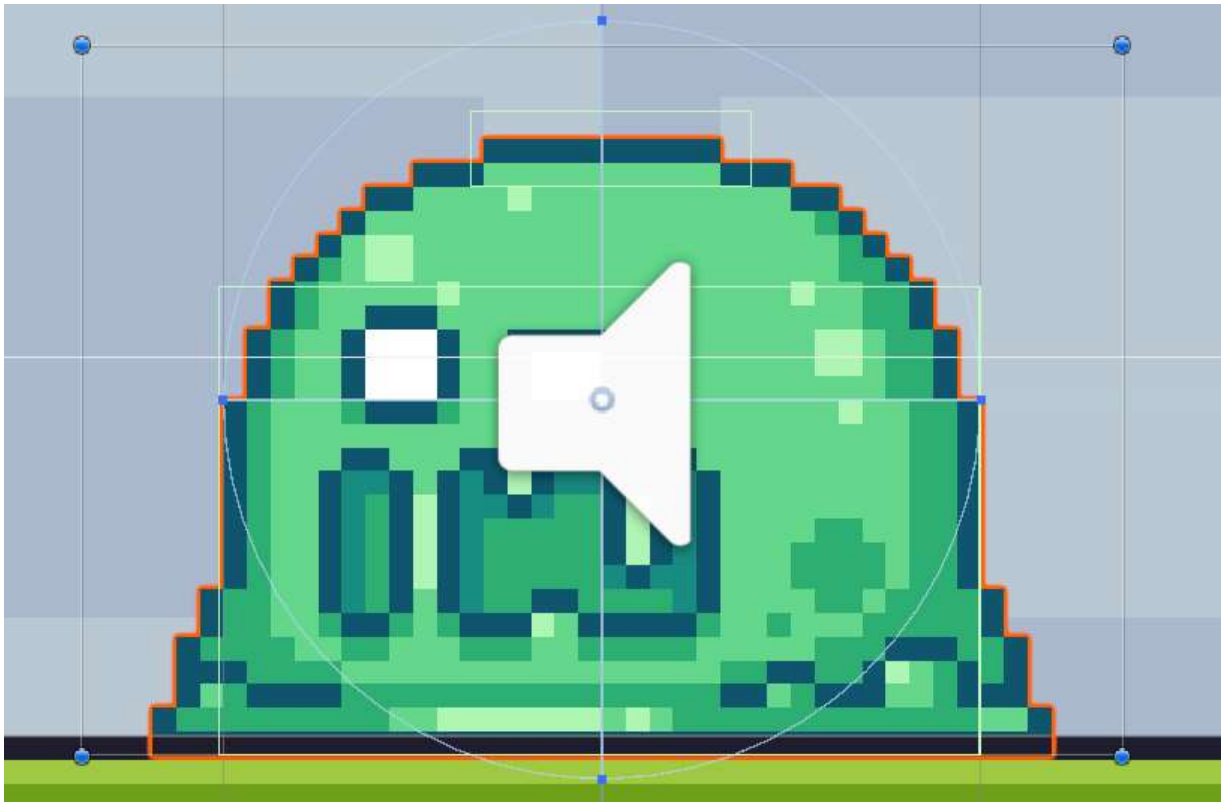


Can change drop force in the editor

Slime

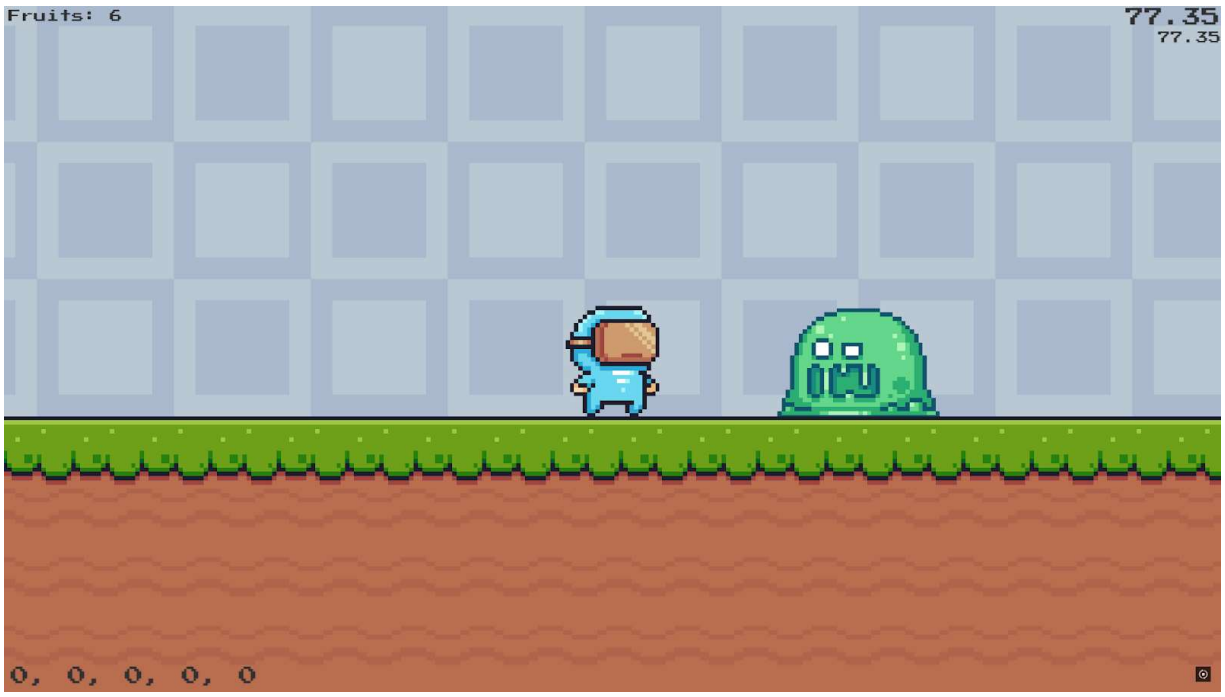
Player dies if they touch the side of the slime, but if they jump on top of the slime they bounce and kill the slime. Slimes have the following components:

- Sprite renderer
- Animator
- Trampoline
- Box collider 2D
    - 1 for collision
    - 1 for trigger
- Audio source
- Enemy death



Slime and box colliders

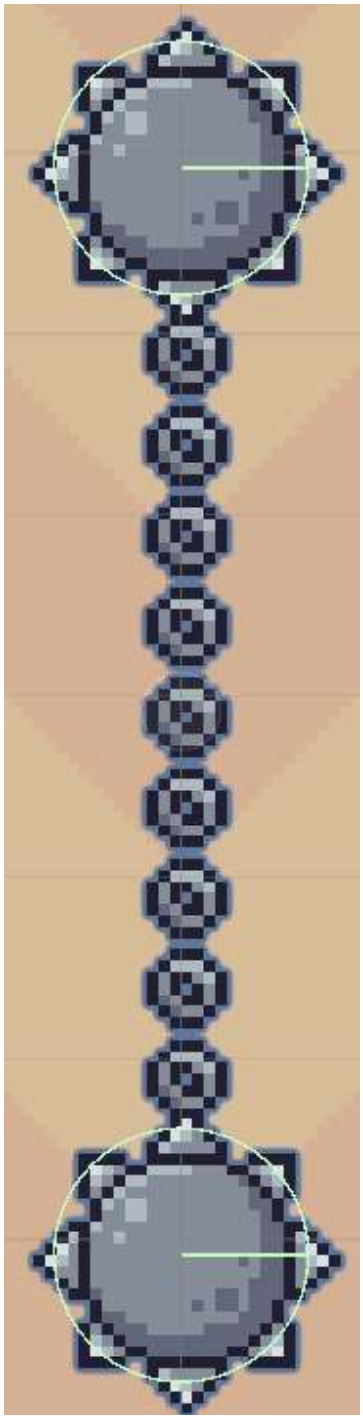

Fruits: 5
67.41
67.41
0, 0, 0, 0, 0


Fruits: 6
77.35
77.35
0, 0, 0, 0, 0

Slime killed when jumped on

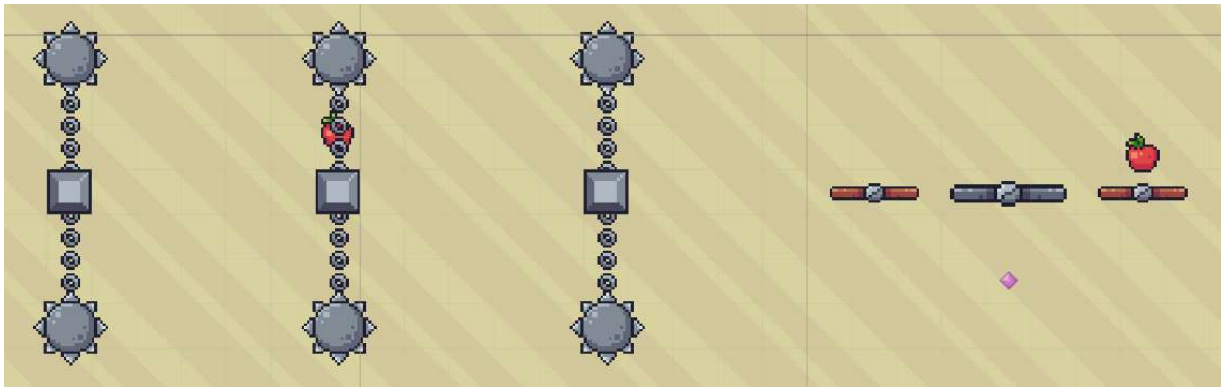
Rotating Spike Balls

Two spike balls held together rotate. The player dies if they collide with the spike balls. The parent game object has the rotate script, which rotates the chains and balls collectively. Each chain and spike ball had a sprite renderer, and each spike ball has a circle collider.

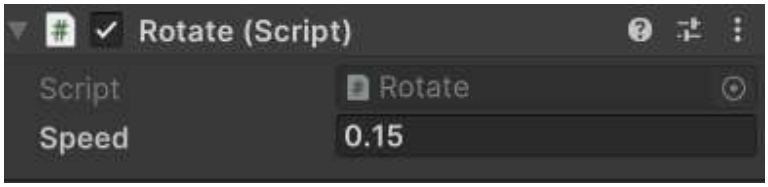


Rotating spike balls and circle colliders

Level 3 rotating spike ball section



Level 5 rotating spike ball section



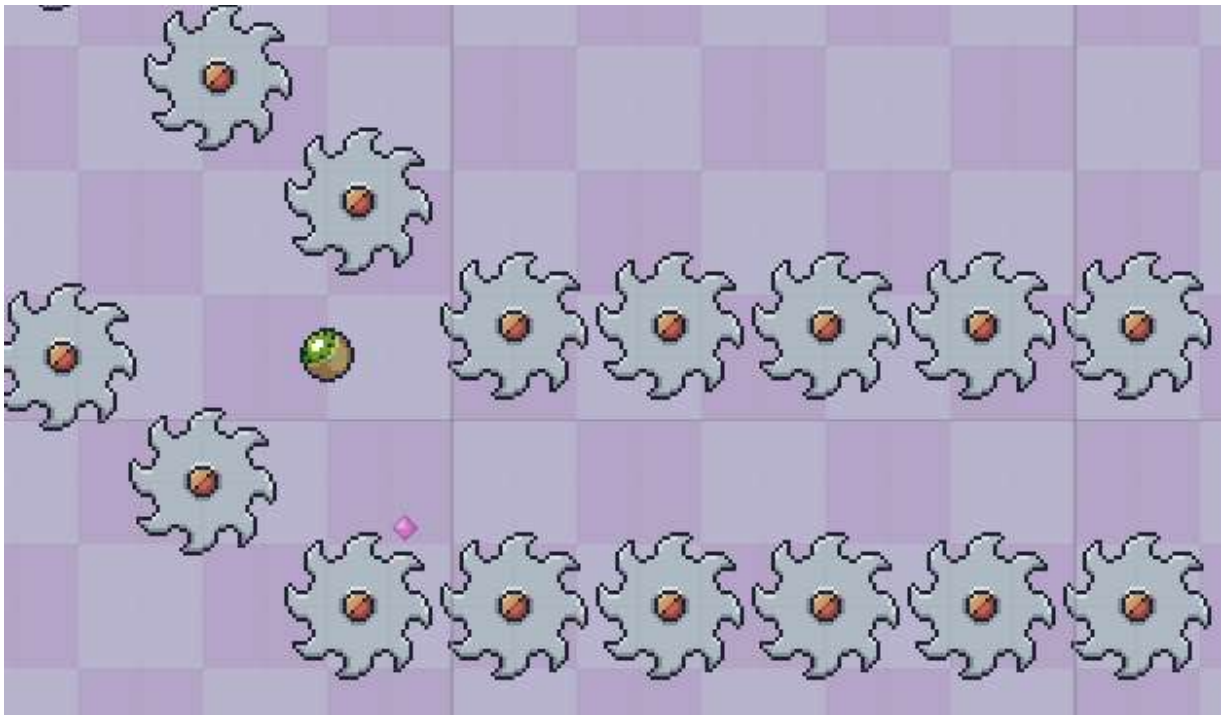
Rate of rotation can be changed in the editor (negative speed changes direction)

Saws

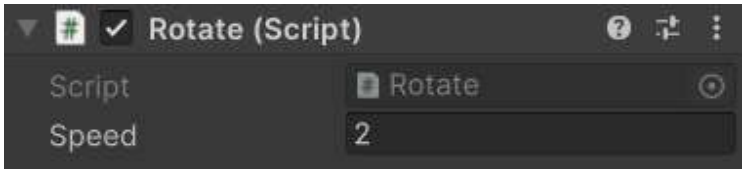
Circular saws rotate and kill the player. They can either be stationary or move along waypoints. They have the following components:

- Sprite renderer
- Waypoint follower (not needed for stationary saws)
- Rotate
- Circle collider 2D

Saws and waypoints



Collection of stationary saws



Rotation speed can be adjusted in the editor (negative values allow for reverse rotation)

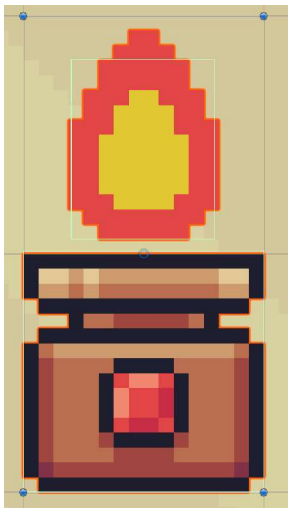
Fire

Player dies if they collide with the flame, but the base of the flame acts the same as normal ground. Fire has the following components:

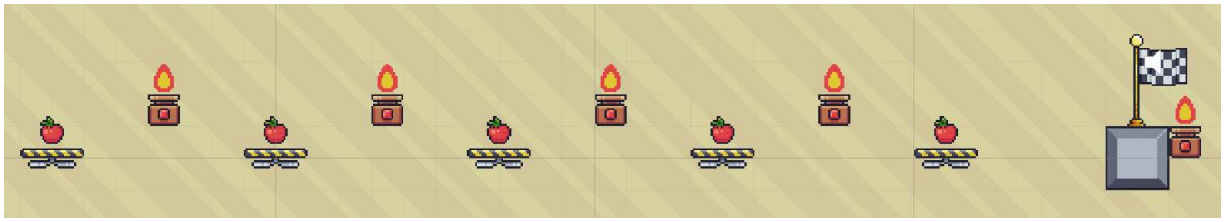
- Sprite renderer
- Animator
- Box collider 2D

There is also a separate gameobject for the flame itself, which just contains a box collider 2D for the flame.

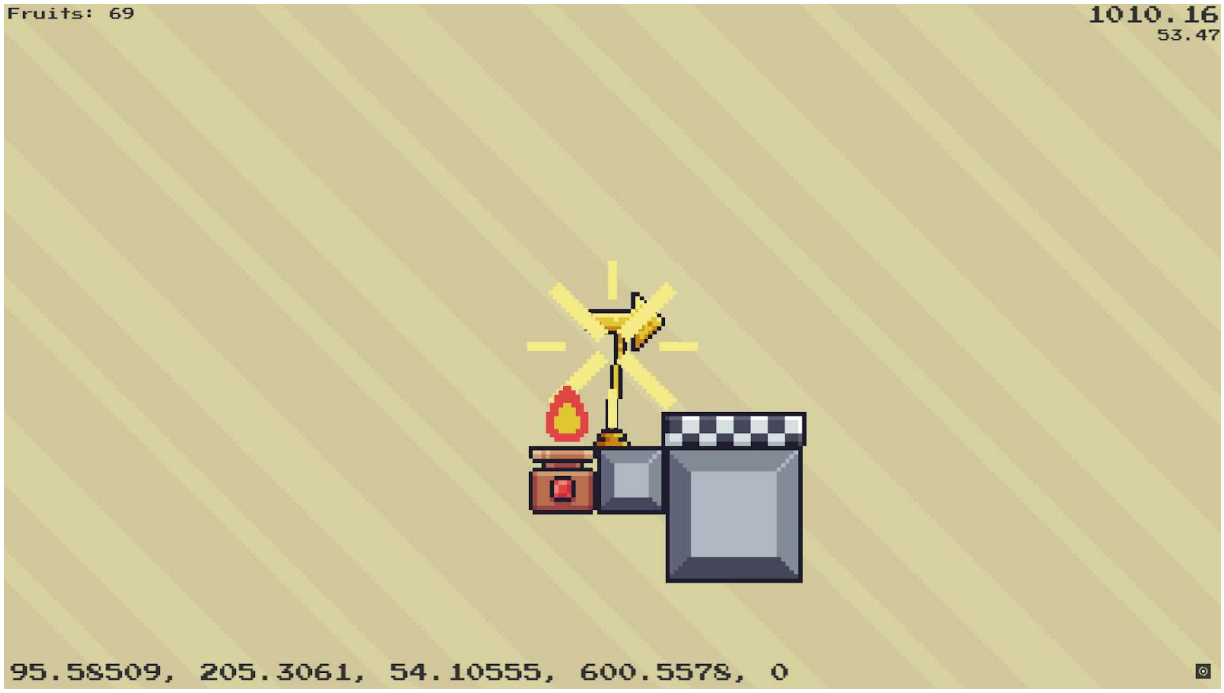
Fire hierarchy



Fire and its box colliders



Fire section in level 5

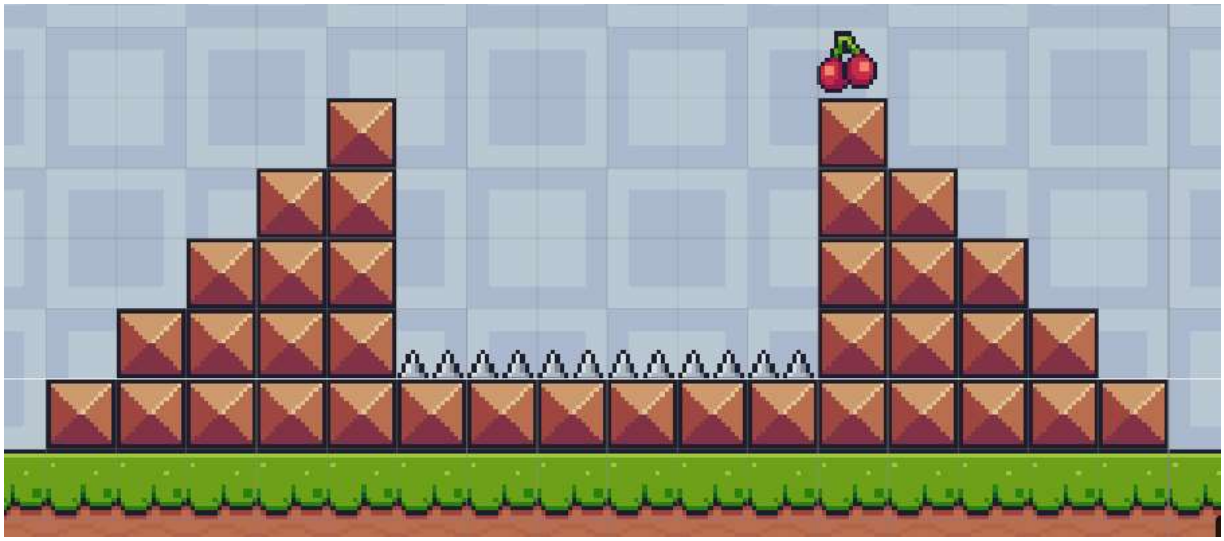


Fire kills player

Spikes

Spikes are simple traps that kill the player on collision. They have the following components:

- Sprite renderer
- Box collider 2D



Spike pit

Arrows

Arrows point the player in the right direction. They can't be interacted with. They have the following components:

- Sprite renderer
- Animator

Arrow

Flying Item

Replicates the behaviour of the cloud in Super Mario World. The player can enter it and fly around (player not visible because it is behind the flying item in the z plane). It has the following components:

- Sprite renderer
- Flying item
- Sticky platform
- Animator
- Box collider 2D
    - 3 for collision
    - 1 for trigger box
- Rigidbody 2D

The propellor is a child of the Flying Item game object and just has animator and sprite renderer components.

Flying item. Trigger zone is in the middle where the player will take control of the flying item upon trigger enter. Box colliders for outer walls and floor of the flying item.

Trampoline/Fan

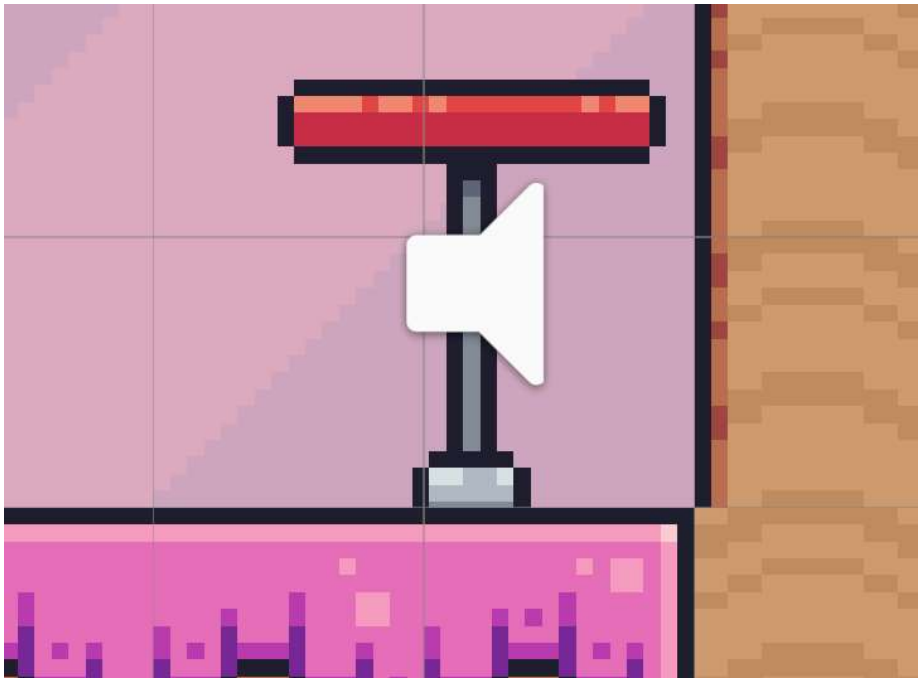
Trampolines and fans launch the player up by adding to the y velocity vector of the player's rigidbody2D. They have the following components:

- Sprite renderer
- Animator
- Box Collider 2D
    - 1 for collision
    - 1 for trigger
- Trampoline
- Audio source

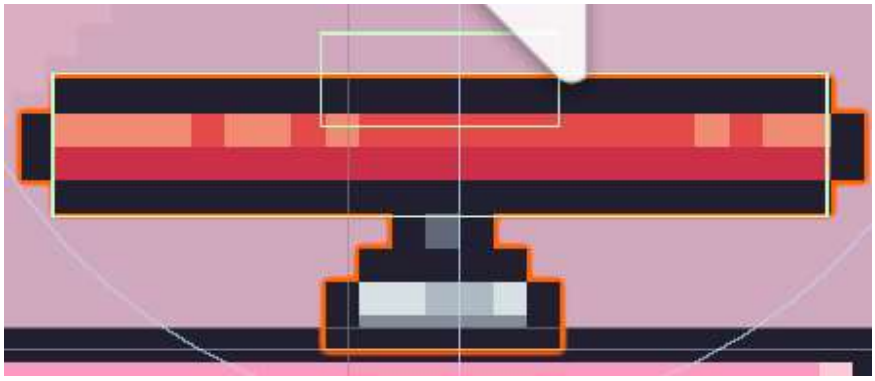
```
C/C++
rb.velocity = new Vector2(rb.velocity.x, jumpForce);
```

Vector2 adds to y velocity of player's rigidbody2D

Trampoline



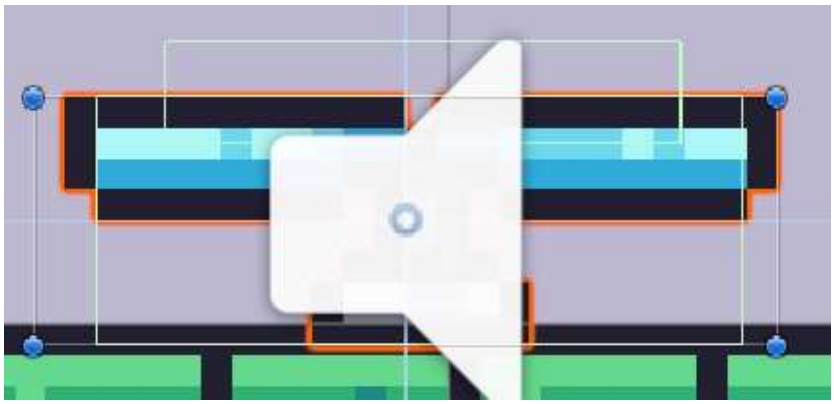
Trampoline extended



Trampoline box colliders

Trampoline editor options

Fan



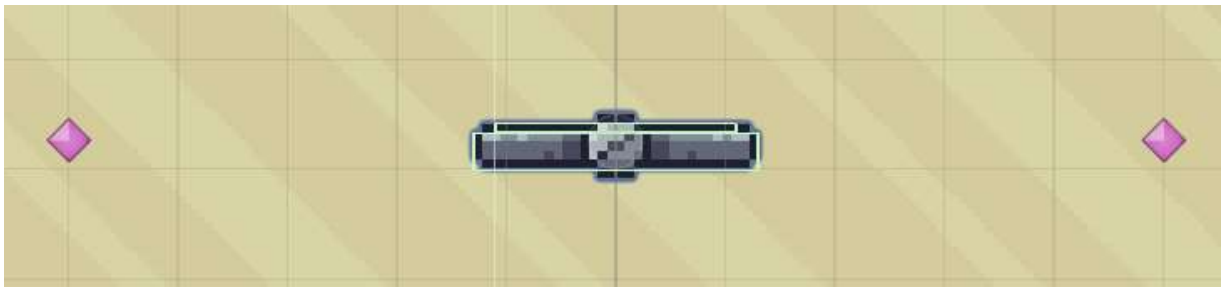
Fan box colliders

Fan has a higher jump force than trampoline (changed in editor)

Moving platform

The moving platform carries the player between waypoints. It has the following components:

- Sprite renderer
- Animator
- Waypoint follower
- Box collider 2D
    - 1 for collision
    - 1 for trigger
- Sticky platform

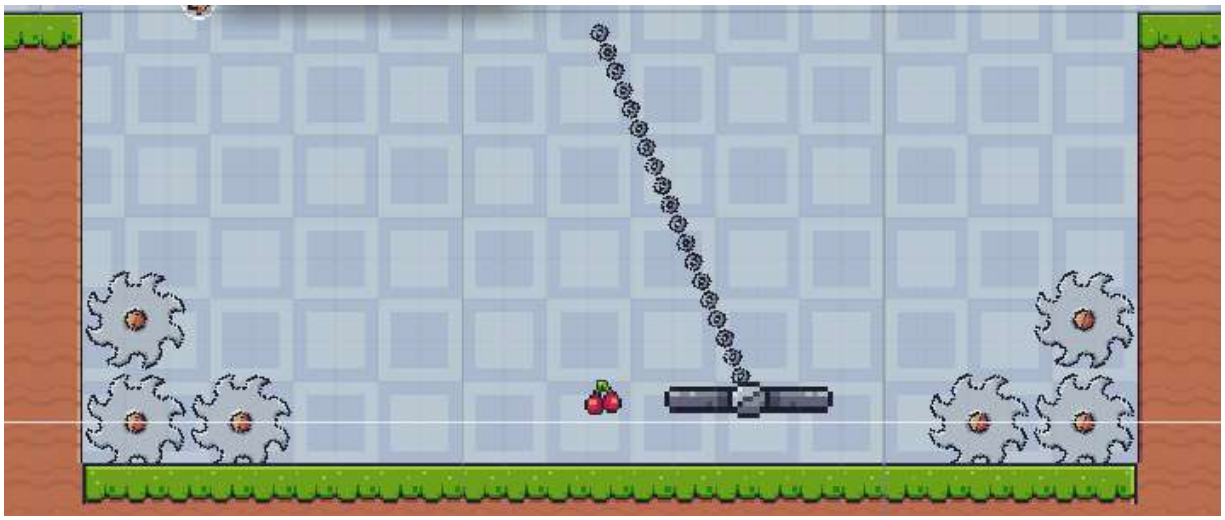


Moving platform and its waypoints.

Swinging platform

The swinging platform has two children. The chains have the swing script (each chain has its own sprite renderer). There is a moving platform that follows the end chain as its only waypoint.

Swinging platform



Mid swing

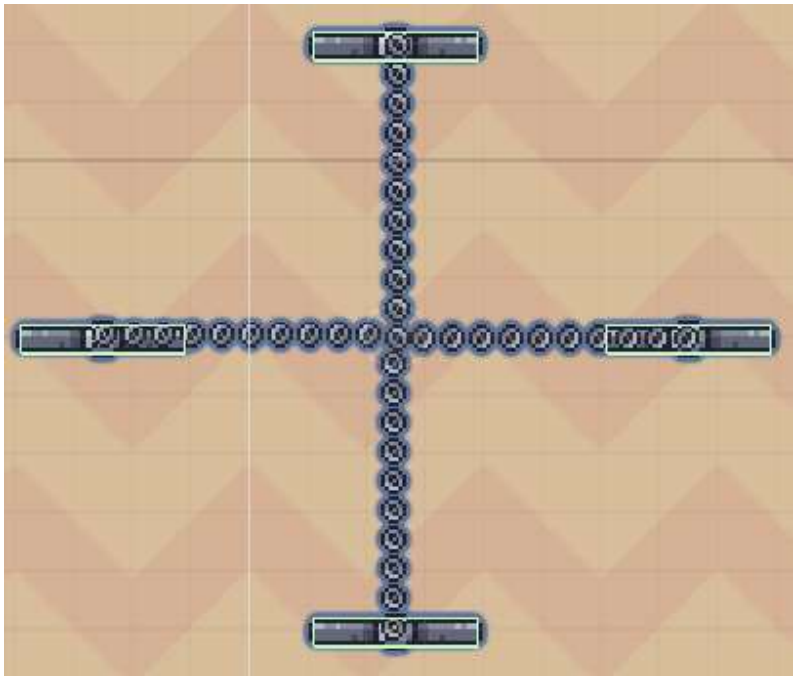
Editor view

Rotating platform

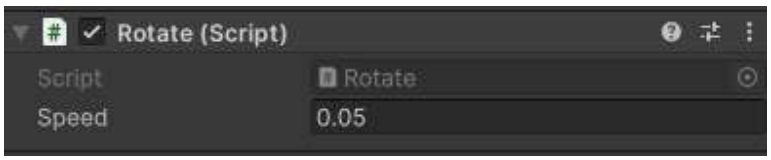
The rotating platform has 6 children:

- 4 moving platforms which follow the end chain of each branch as their only waypoint
- Centre chain which is the centre of rotation
- Rotating chains

Seen in editor



Rotating platform

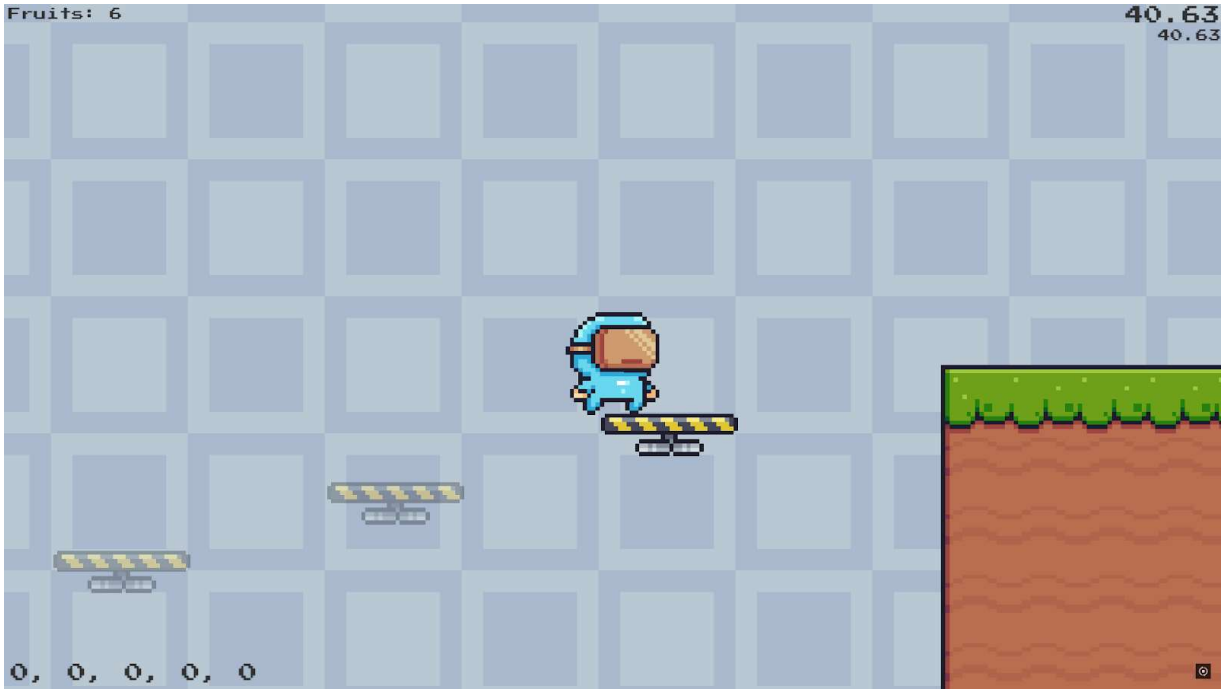


Can adjust speed of rotation in the editor (reverse value allows for reverse rotation)

Recurring platform

Recurring platform disappears x seconds after being stepped on, and returns y seconds later. It has the following components:

- Sprite renderer
- Animator
- Box collider 2D
- Recurring platform



Platforms disable (left) x seconds after first being stood on and return to active (right) y seconds later.

On Off platforms

Platforms alternate between on and off states. They have the following components:

- Sprite renderer
- Box collider 2D
- Animator
- On Off Platform

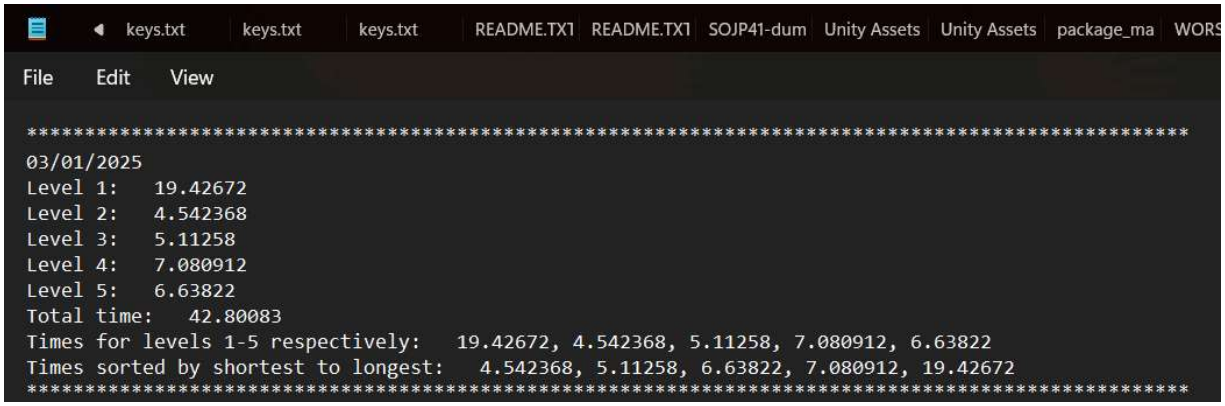
Platforms alternate between off (left/right) and on (middle)

Values can be adjusted in the editor to change time in each state and delay to offset alternating platforms from each other.

Times Text File

Times will be written to a text file on the desktop when the end screen is reached (so does not right if the game is quit). The text file will be able to be opened manually in a text editor, but I will explore opening it in game. Times are displayed to 6 decimal places to be as accurate as possible.

Early time formatting for each playthrough.



Final time formatting for each playthrough. Added array output including array sorted by merge sort algorithm.

```C/C++
private void SaveTimesToFile()
{
    //fileWritten bool ensures file is only appended to once per game
completion
    if (!fileWritten)
    {
        //Adds line
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**************************************************************************
**********************" + Environment.NewLine);
        //Appends todays date
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
DateTime.Today.ToShortDateString() + Environment.NewLine);

        //Appends each level number and time by iterating with a for loop
        for (int i = 0; i < 5; i++)
        {
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Level "
+ (i + 1).ToString() + ":   " + times[i].ToString() + Environment.NewLine);
        }

        //Appends total (global) time
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Total time:
" + currentTime.ToString() + Environment.NewLine);

        //
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times for
levels 1-5 respectively:   " + string.Join(", ", times) + Environment.NewLine);
        //
        sortedTimes = MergeSort(times);

        //
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times
sorted by shortest to longest:   " + string.Join(", ", sortedTimes) +
Environment.NewLine);

        //Adds line
        File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**************************************************************************
**********************" + Environment.NewLine);
        //Ensures file is not appended to again
        fileWritten = true;
    }
}
```

(Above) Function in Timer script to save times to text file at the end of the game.

```
C/C++
//Merge sort function
private static float[] MergeSort(float[] array)
{
    //Checks if array is just 1 element long or empty
    if (array.Length <= 1)
    {
        return array;
    }

    //Calculates mid point
    int midPoint = array.Length / 2;

    //Creates left and right half arrays
    float[] left = new float[midPoint];
    float[] right = new float[array.Length - midPoint];

    //Populates left and right arrays with respective contents from array
    Array.Copy(array, 0, left, 0, midPoint);
    Array.Copy(array, midPoint, right, 0, array.Length - midPoint);

    //Recursively sort left and right halves until the array is completely
sorted
    left = MergeSort(left);
    right = MergeSort(right);

    //Sort left and right halves using Merge function and return sorted array
    return Merge(left, right);
}

//Merges two arrays
private static float[] Merge(float[] left, float[] right)
{
    float[] result = new float[left.Length + right.Length];

    //Position in left array
    int i = 0;
    //Position in right array
    int j = 0;
    //Position in "array" array
    int k = 0;
```

```
    //Compare elements from the left and right arrays and merge them in sorted
order
    while (i < left.Length && j < right.Length)
    {
        //Comparison
        if (left[i] <= right[j])
        {
            //If left array value is smaller, it is written to
"array" array
            result[k++] = left[i++];
        }
        else
        {
            //If right array value is smaller, it is written to
"array" array
            result[k++] = right[j++];
        }
    }
```

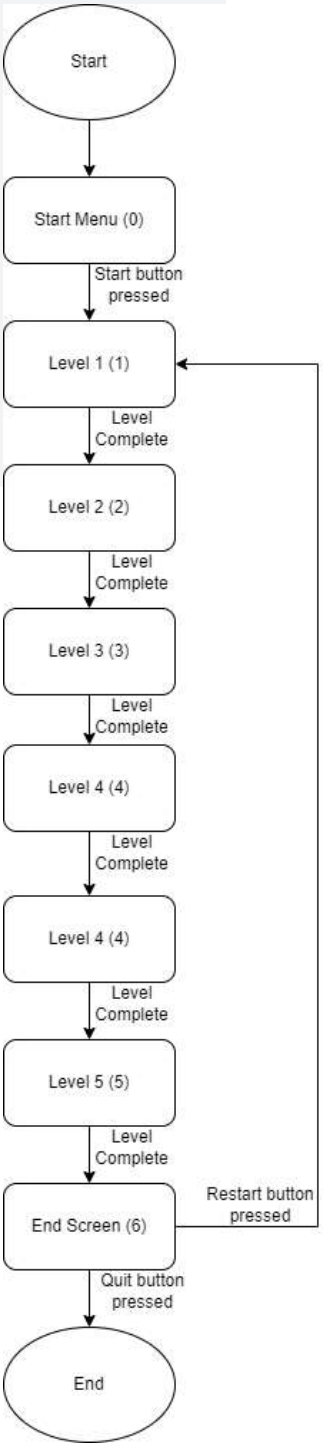
Merge sort functions in Timer script. Base case of the merge sort is when a list has 0 or 1 elements. Time complexity of merge sort is O(n log n) so it is very efficient.

Whole System

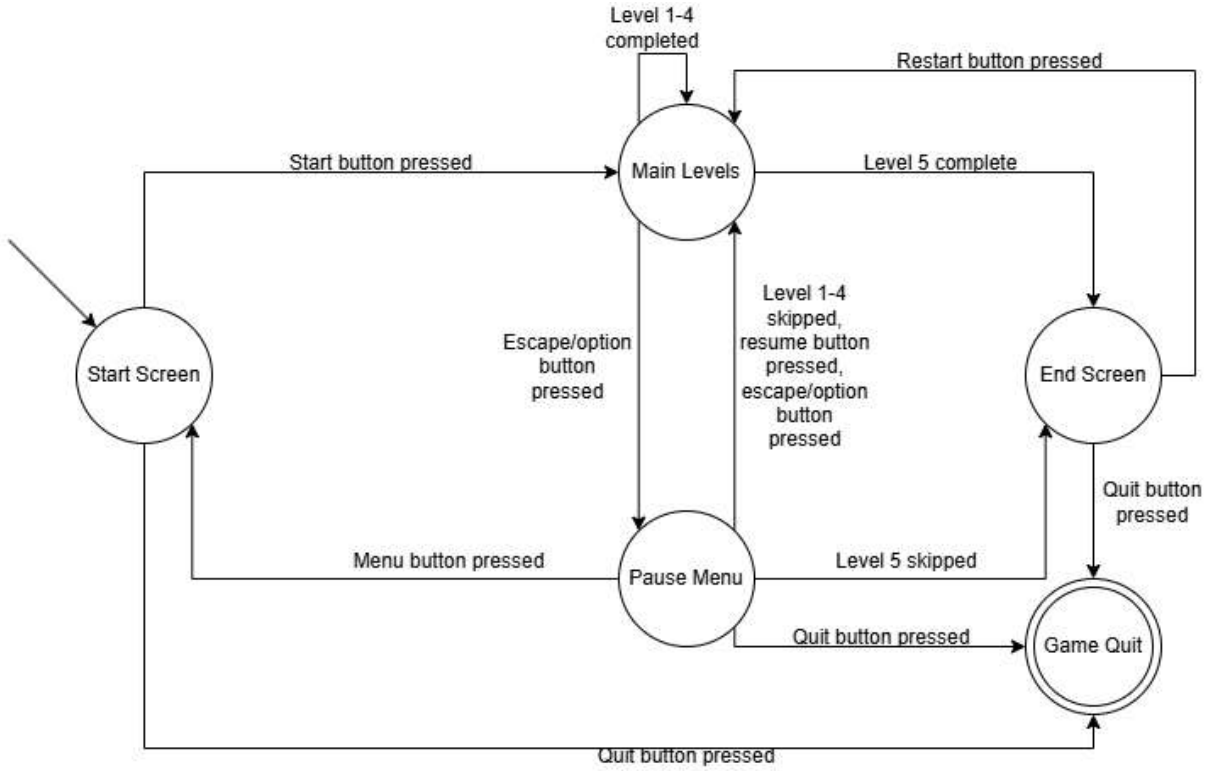
Scene Management

Flowchart

Flowchart shows the order in which scenes are loaded and how to trigger the next scene during gameplay. The number in brackets is the build index of a particular scene, indexes from 0.

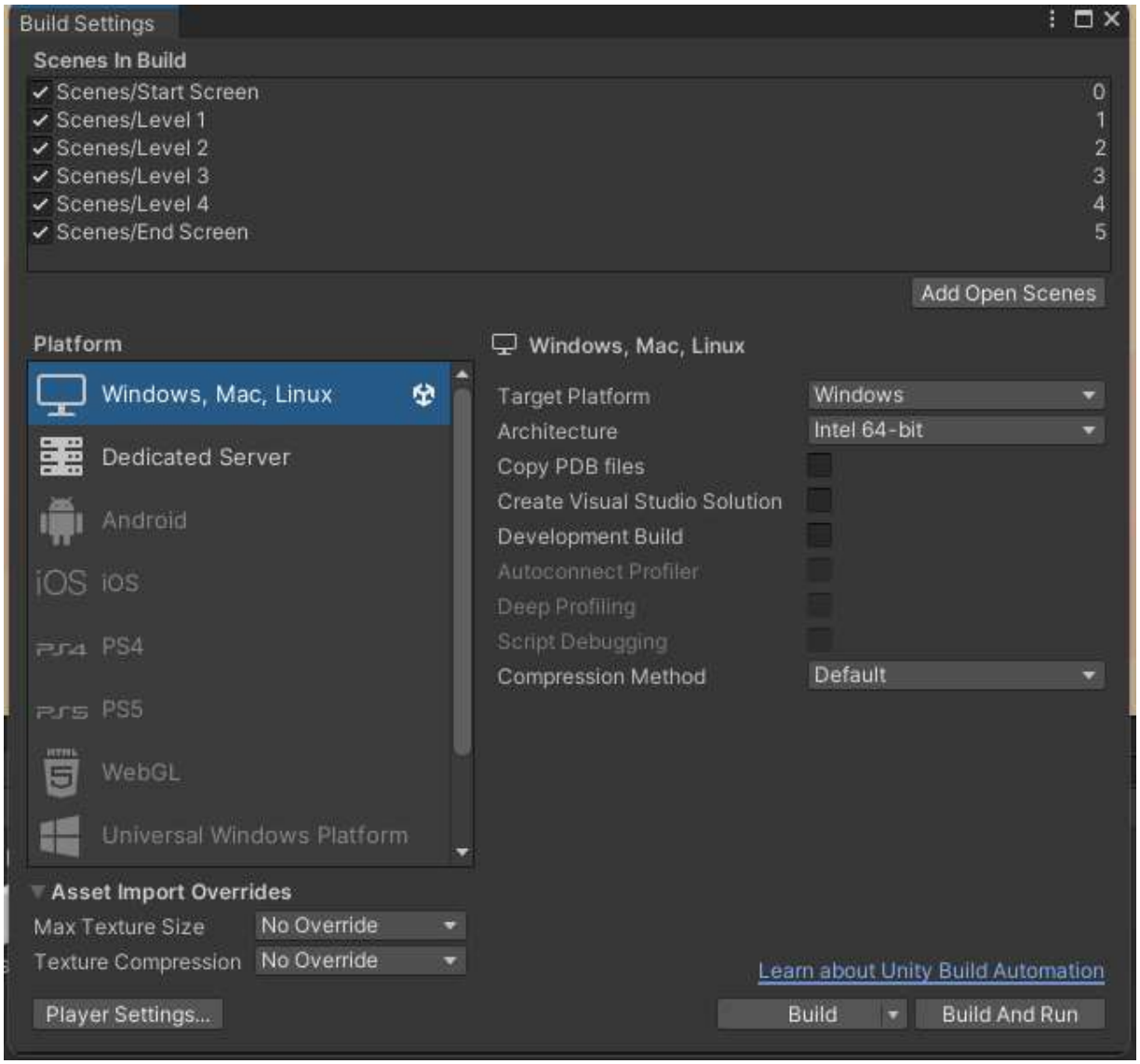


Finite state machine

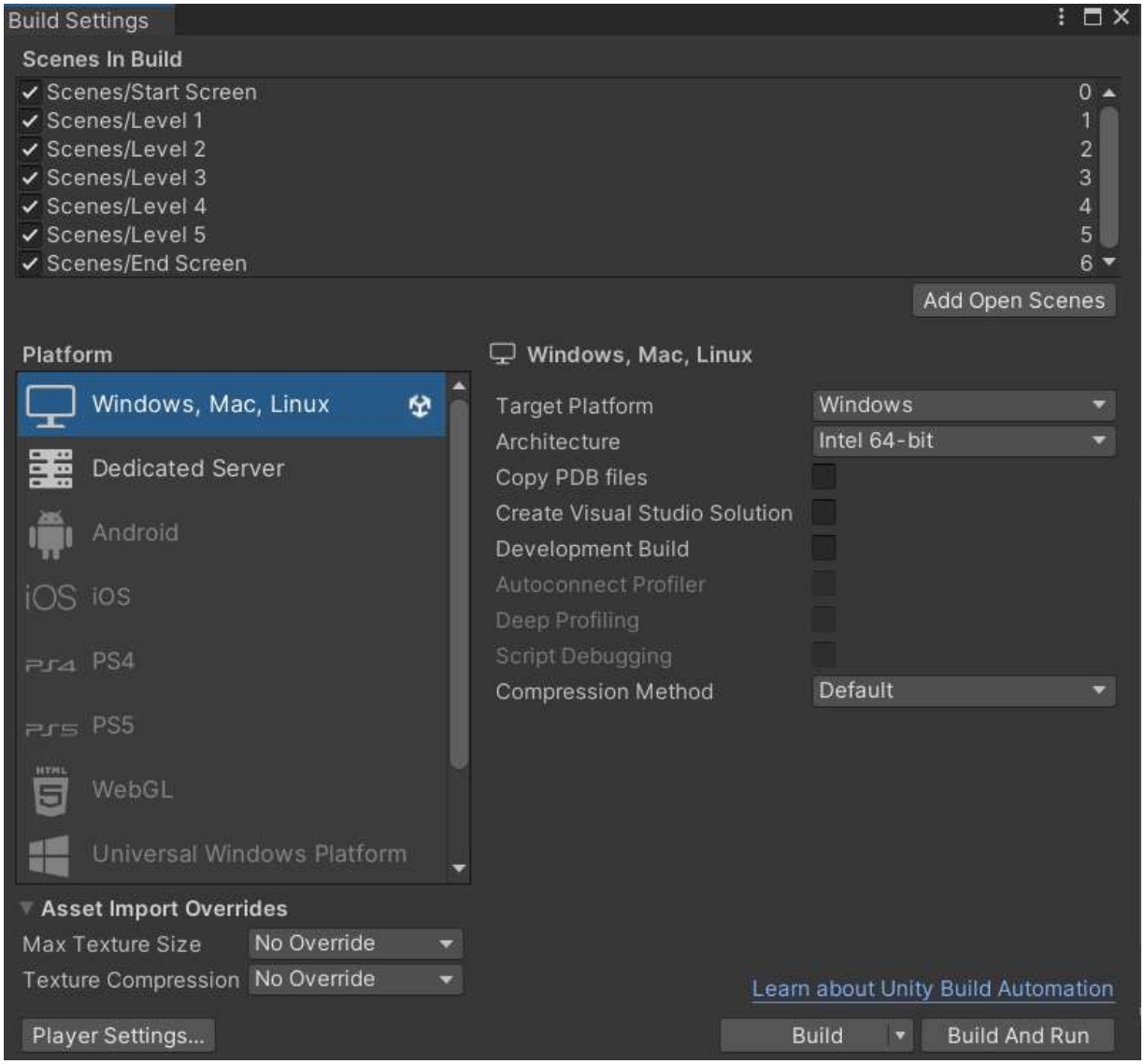


Finite state machine has been used instead of a flowchart because it shows more relations which is needed because the addition of the pause menu allows for many more changes between scenes and UI changes. This should allow for any combinations of user gameplay choices.

Build Settings



Early build settings window (before level 5 added).

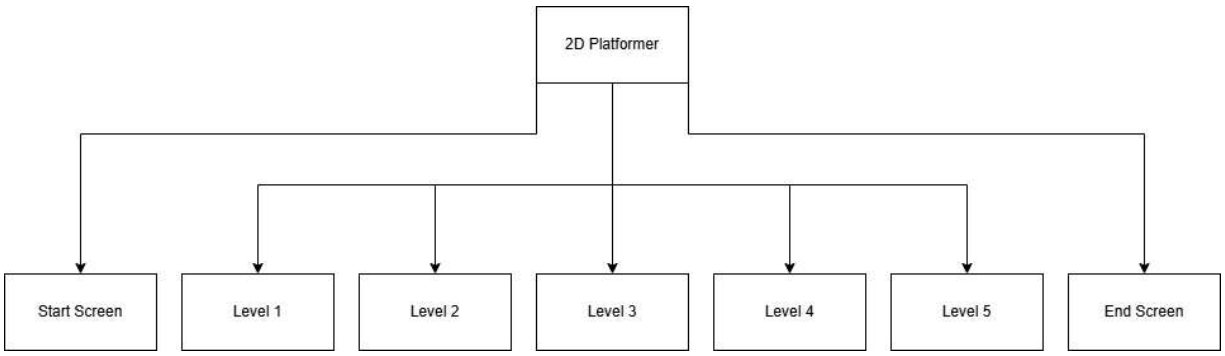


Final build settings (after level 5 added).

The build manager loads the next scene when the start button is pressed (for the start menu) or 2 seconds after the finish line is triggered, in the StartMenu and Finish scripts respectively. The restart button loads scene 1 (Level 1). I will build for 64-bit windows as that is the operating system I am running (Windows 11). Builds/compiles an .exe executable.

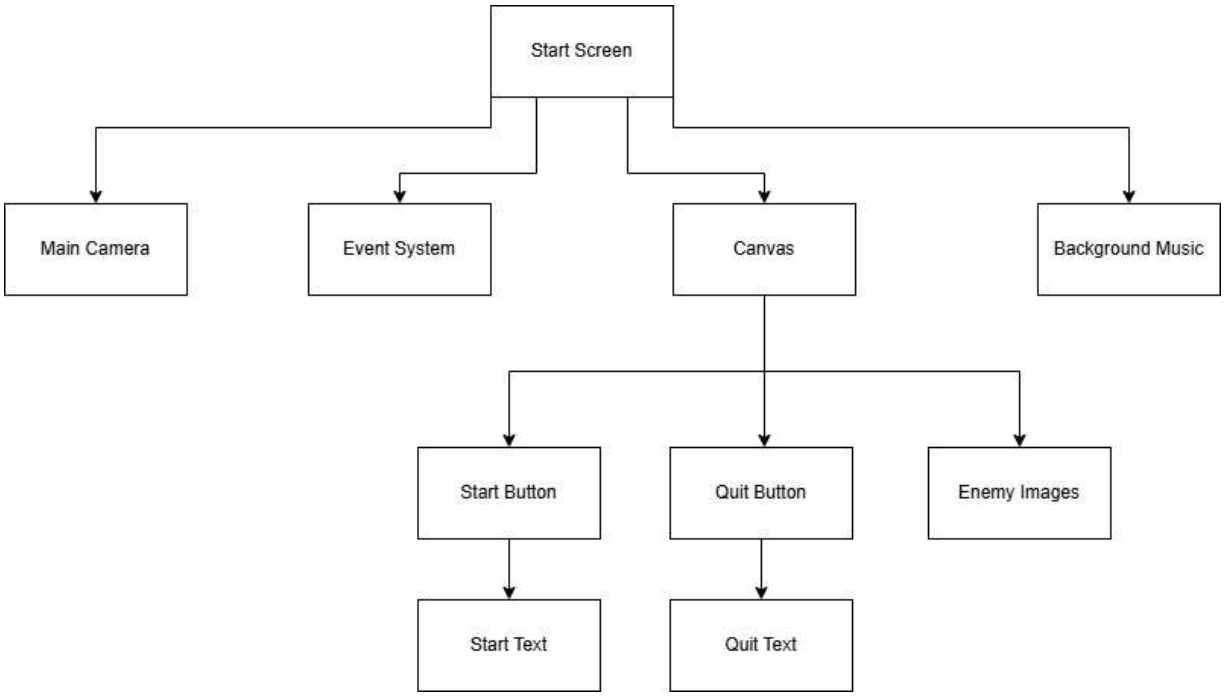
Hierarchy diagrams

Scene hierarchy



Flat hierarchy diagram shows scenes have no priority over each other. Each scene is loaded as shown in the finite state machine in the previous section.

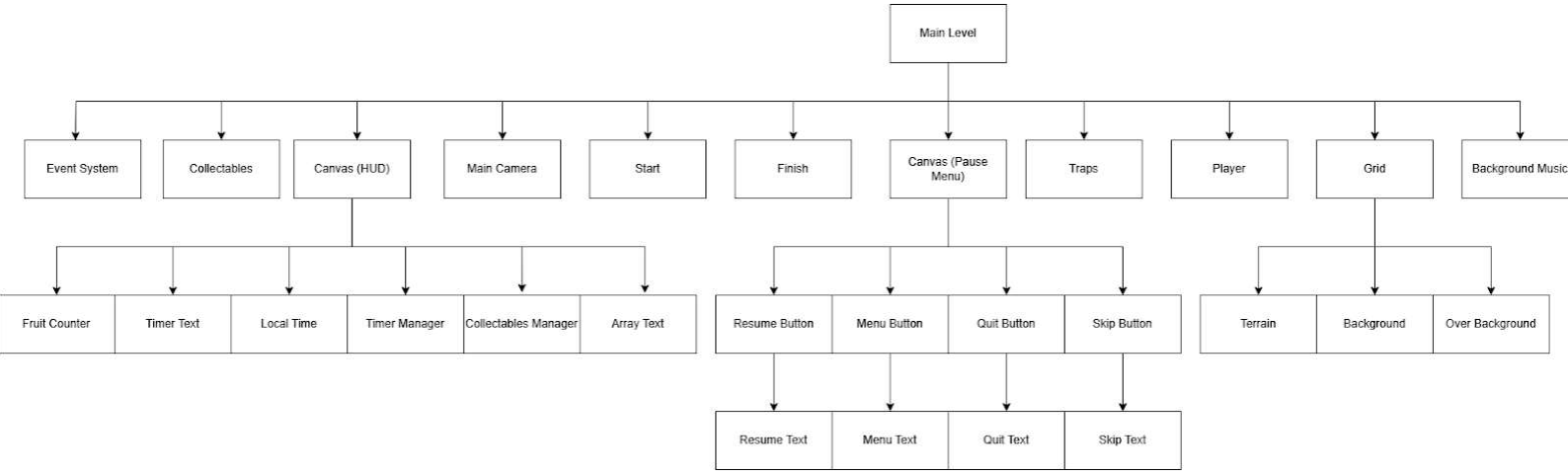
Start screen hierarchy



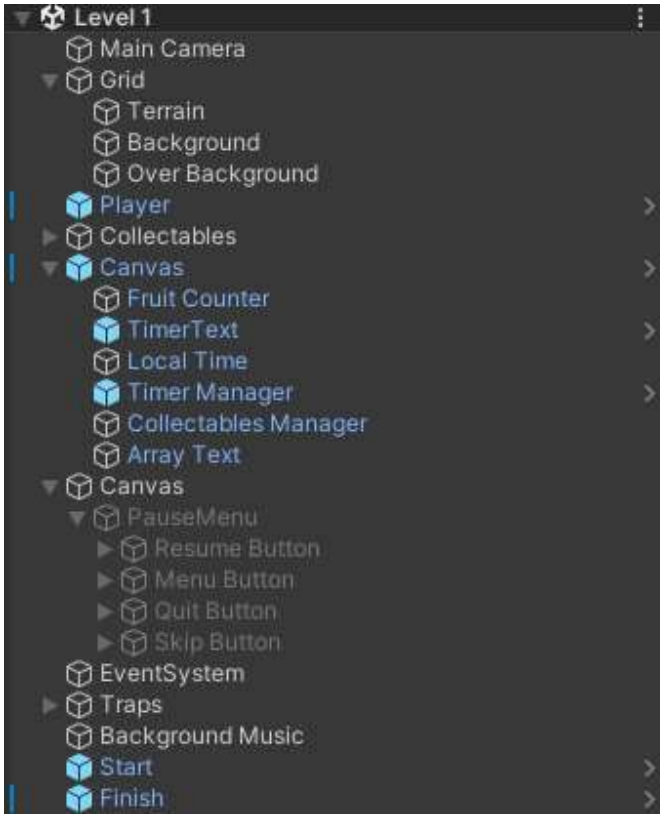
Shows the game object hierarchy under the Start Screen scene.

Start Screen game object hierarchy seen in the editor.

Main levels hierarchy

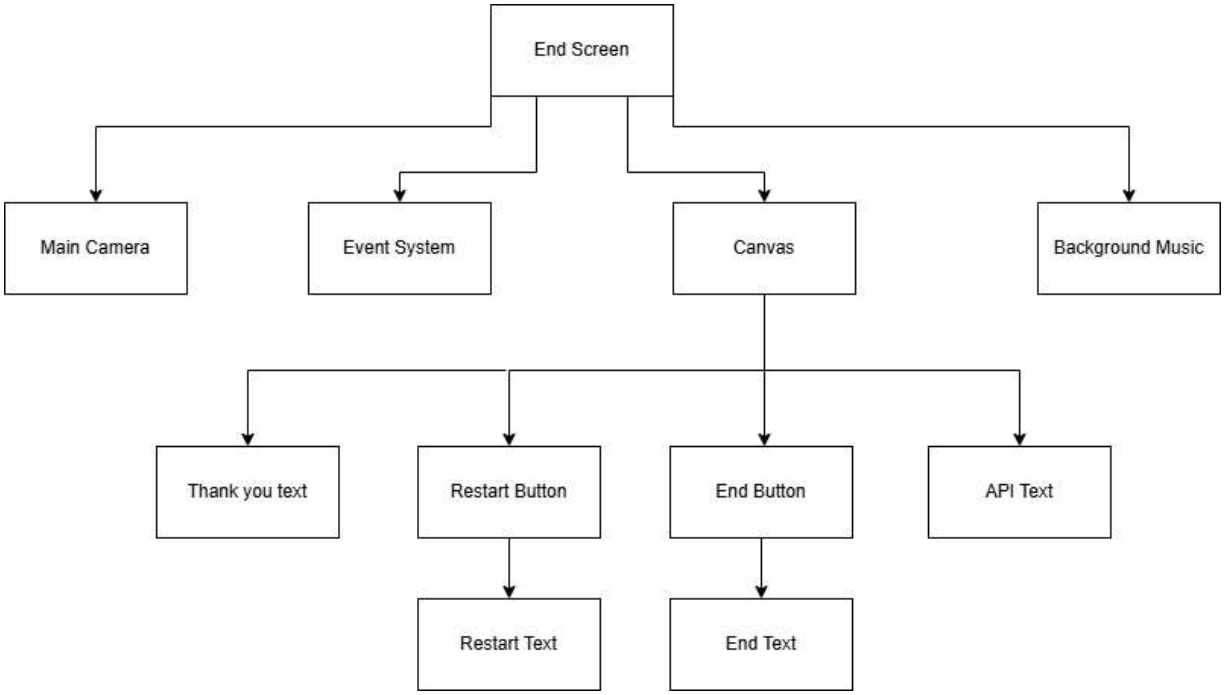


This shows the hierarchy of game objects under any main level scene. “Traps” and “Collectables” also contain children game objects but have been discussed in more detail elsewhere, as a diagram doesn’t show enough detail.

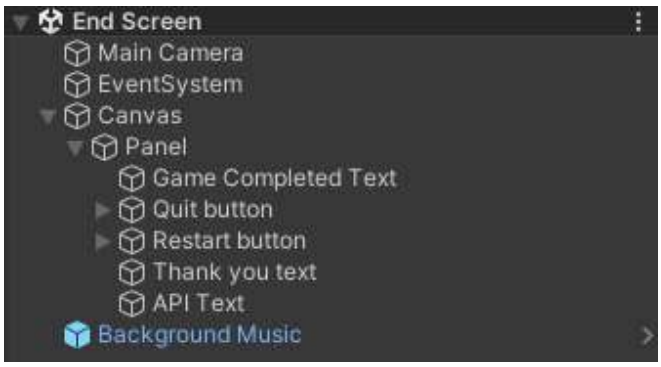


Game object hierarchy of level 1 in the editor.

End screen hierarchy



Shows the game object hierarchy under the End Screen scene.

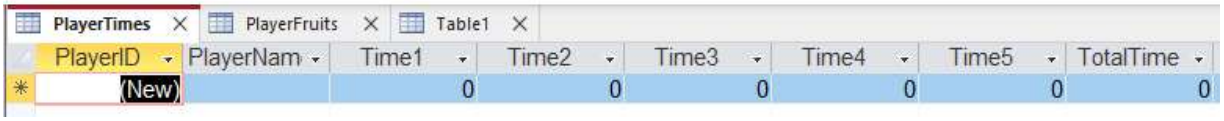


EndScreen game object hierarchy seen in the editor.

SQL Databases*

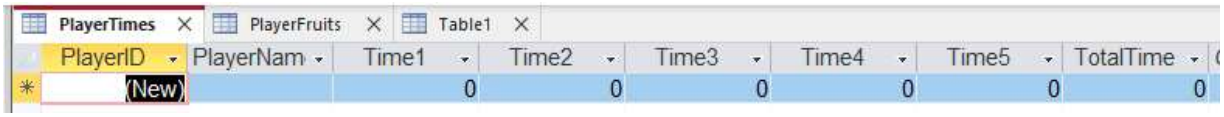
I could to implement tables for the times and number of fruits collected by the player. This would contain both the individual values for each level, and the whole table. All tables would use a primary key “PlayerID” to identify which values apply to which player.

PlayerTimes



Early table design in Microsoft Access.

PlayerFruits



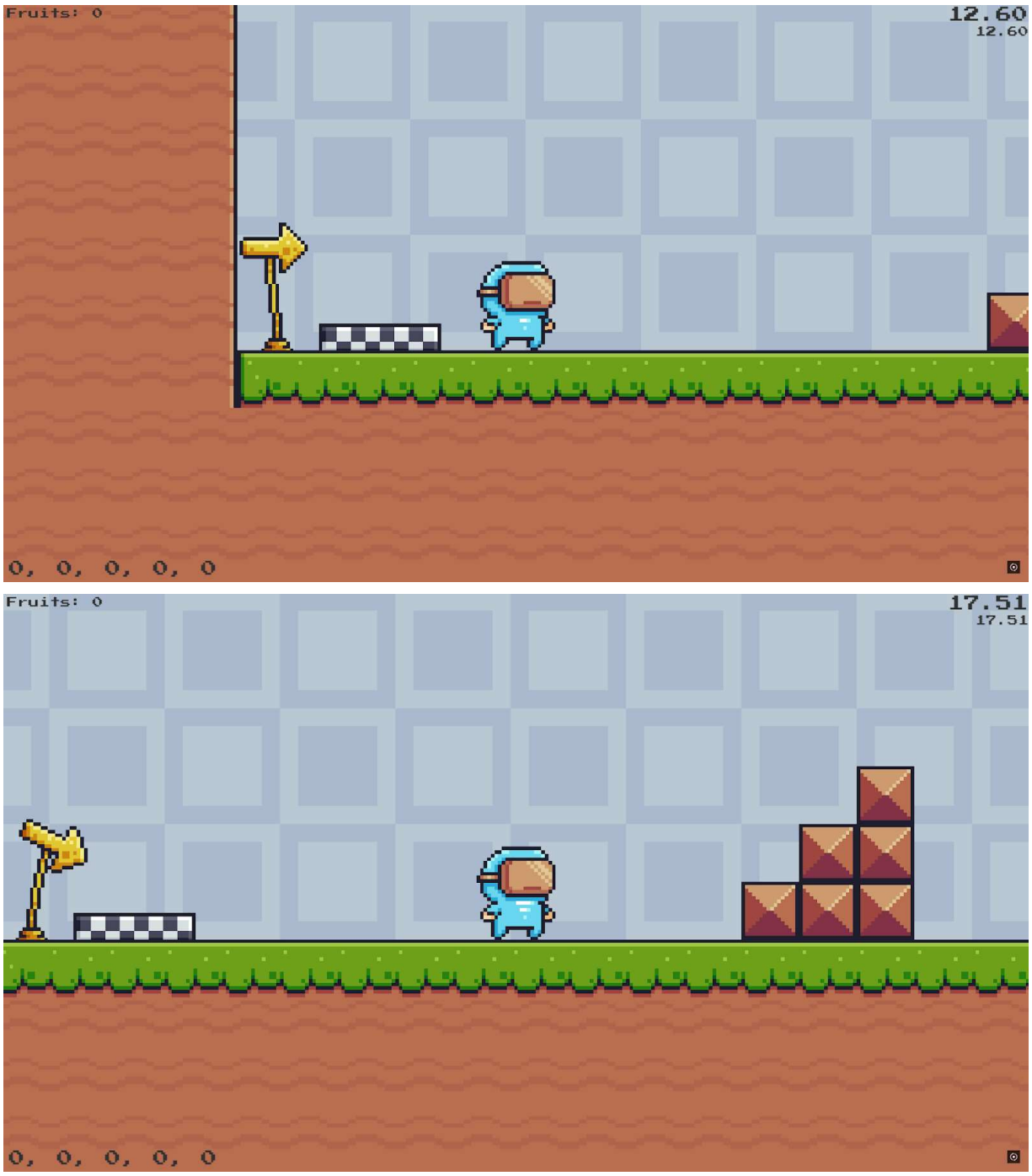
Early table design in Microsoft Access.

* Not used in the final system. I decided against implementing SQL in favour of using text files instead because they are easier to work with and easier for the end user to access directly.

Technical Solution

Camera Controller

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraController : MonoBehaviour
{
    //Allows player's transformation/position to be read
    [SerializeField] private Transform player;


    // Update is called once per frame
    private void LateUpdate()
    {
        //Each frame the camera is transformed to same position as player,
keeping different Z value so the player can still be seen
        transform.position = new Vector3(player.position.x, player.position.y,
transform.position.z);
    }
}
```

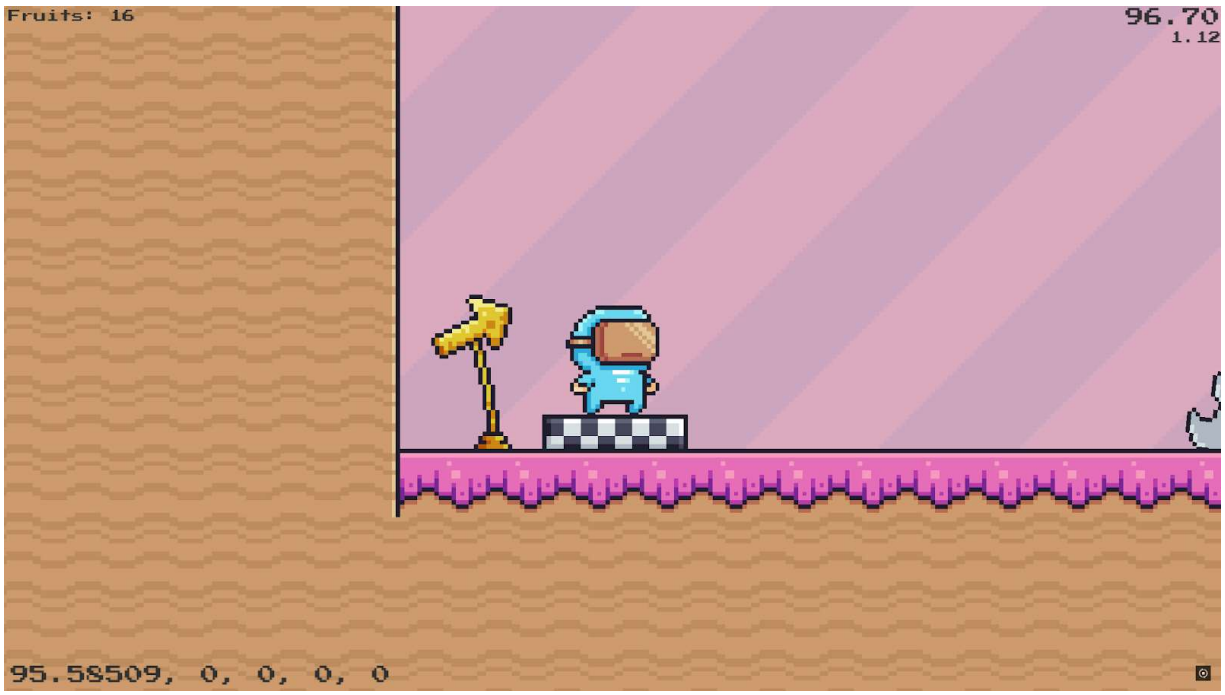


Camera follows player

Collectables Manager

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class CollectablesManager : MonoBehaviour
{
    [SerializeField] private Text FruitTextCounter;
    public int fruits = 0;
    private int savedFruits;

    void Start()
    {
        //0 Fruits collected at start
        fruits = 0;
        savedFruits = 0;
    }

    public void AddFruits(int n)
    {
        //Increments fruit counter when player touches a fruit
        fruits += n;
        FruitTextCounter.text = ("Fruits: " + fruits.ToString());
    }

    public void SaveFruits()
    {
        //Saves the current fruit count so it continues across levels
        savedFruits = fruits;
    }

    public void LoadFruits()
    {
        //Sets fruit counter to the number of fruits at the end of the last
level when a new level is loaded
        fruits = savedFruits;
    }

    //Calls LoadFruits() function when a new level is loaded
    void OnScreenLoaded(Scene scene, LoadSceneMode mode)
```

```
    {
        LoadFruits();
    }

    //Keeps track of when a new scene is loaded
    private void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    //Keeps track of when a level ends
    private void OnDisable()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }

    //Resets fruit counter and its text at the start of level 1
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.buildIndex == 1)
        {
            fruits = 0;
            savedFruits = 0;
            FruitTextCounter.text = ("Fruits: " + fruits.ToString());
        }
    }
}
```



Fruit counter is persistent across levels (here fruits from level 1 are present at the start of level 2)

Disappearing Platform*

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DissapearingPlatform : MonoBehaviour
{
    //Allows time to destroy to be changed from Unity engine
    [SerializeField] private float timeToDestroy = 1f;

    //Detects collision
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            //Calls subroutine DisappearAfterDelay()
            StartCoroutine(DisappearAfterDelay());
        }
```

```
    }

    private IEnumerator DisappearAfterDelay()
    {
        //Waits for 1 second
        yield return new WaitForSeconds(timeToDestroy);
        //Destroys the platform
        Destroy(gameObject);
    }
}
```

*Not used as it was replaced by the recurring platform script.

Can adjust time to destroy in editor

End Menu

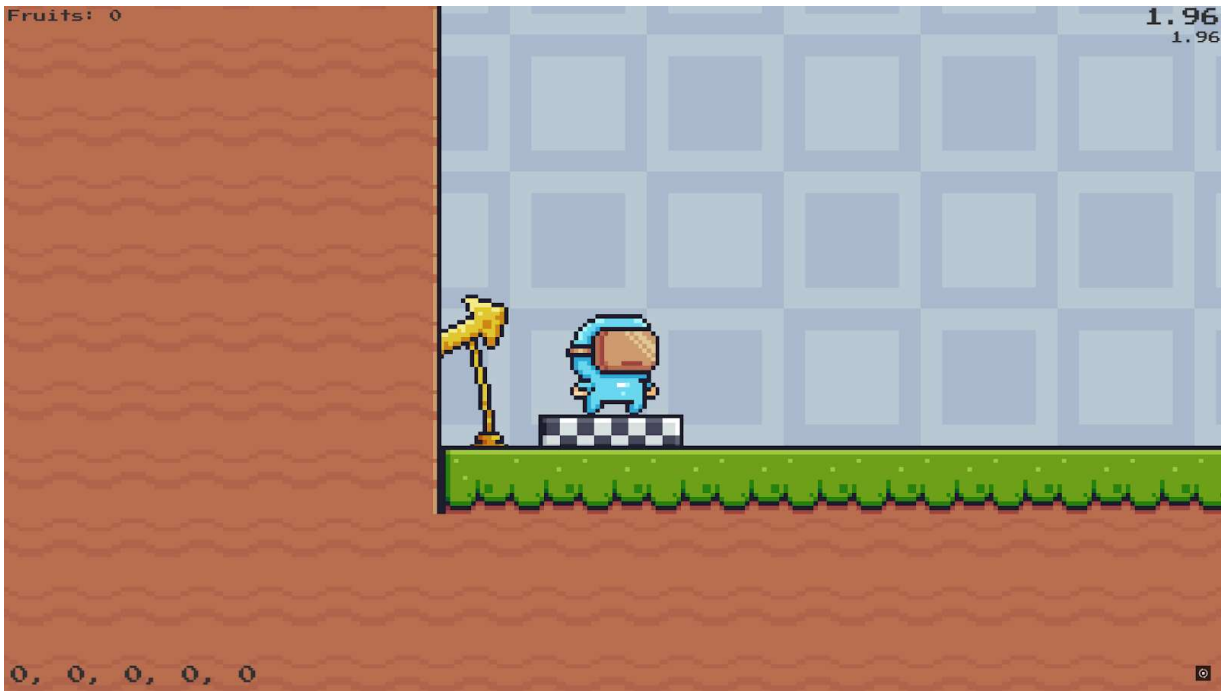
```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class EndMenu : MonoBehaviour
{
    //Public so can be added to on click () in button component
    //Quits the game
    public void Quit()
    {
        Application.Quit();
    }

    //Public so can be added to on click () in button component
    //Restarts the game from level 1
    public void Restart()
    {
        SceneManager.LoadScene(1);
```

```
    }
}
```

Level 5

End screen with API quote (online)

End screen with a random default quote (offline)

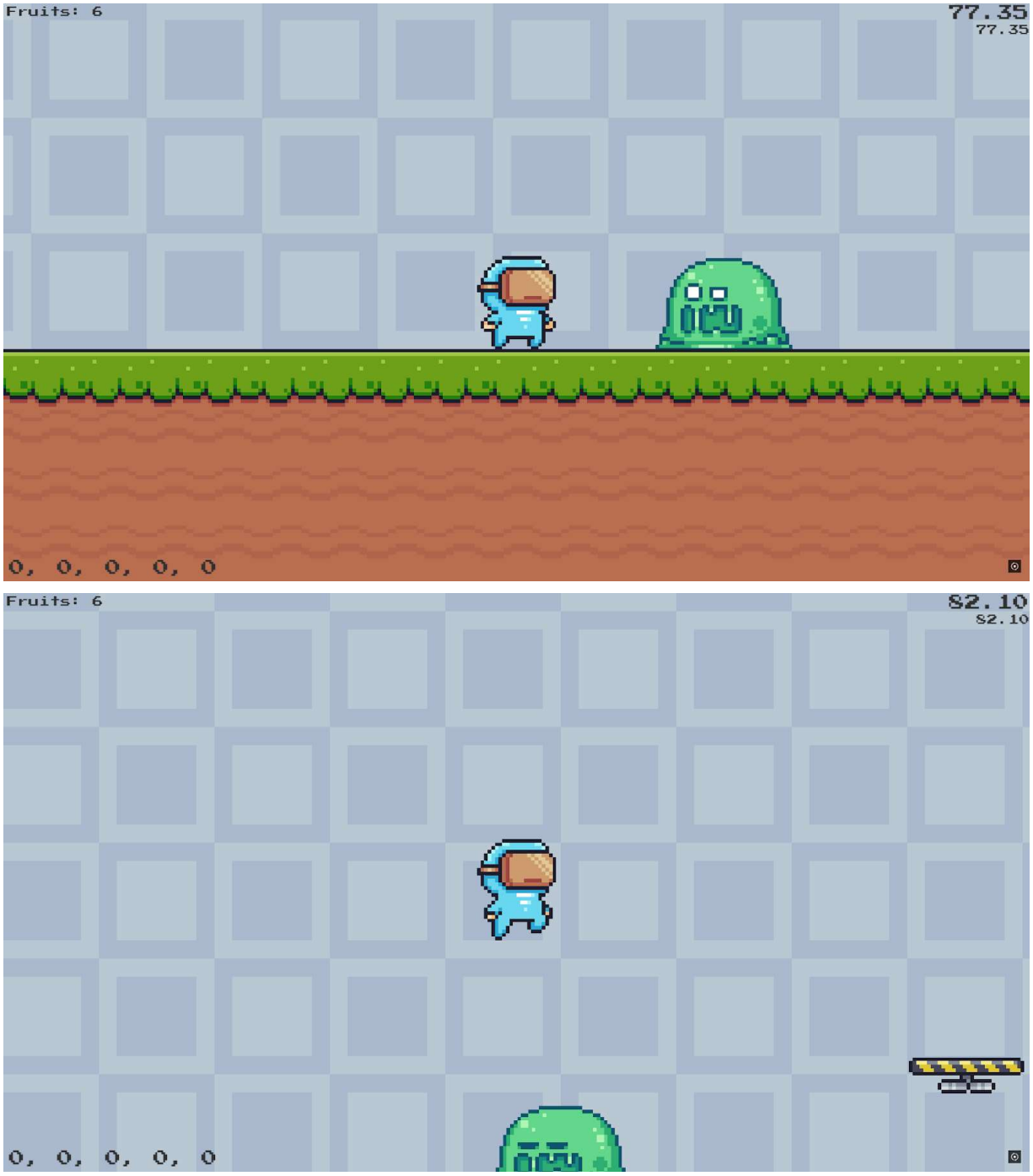


Level 1 loads upon restart

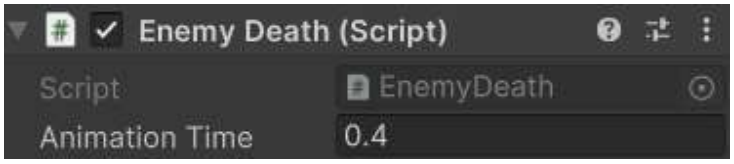
Enemy Death

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyDeath : MonoBehaviour
{
    //Creates Animator variable
    private Animator anim;
    //Allows time for death animation to be input in Unity editor
    [SerializeField] private float animationTime = 1f;

    // Start is called before the first frame update
    void Start()
    {
        //Gets animator component on start
        anim = GetComponent<Animator>();
    }
```

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            //Triggers death animation
            anim.SetTrigger("death");
            StartCoroutine(DisappearAfterDelay());
        }
    }

    private IEnumerator DisappearAfterDelay()
    {
        //Waits for value held by animationTime variable (in seconds) so
animation can play before death
        yield return new WaitForSeconds(animationTime);
        //Destroys the enemy
        Destroy(gameObject);
    }

}
```



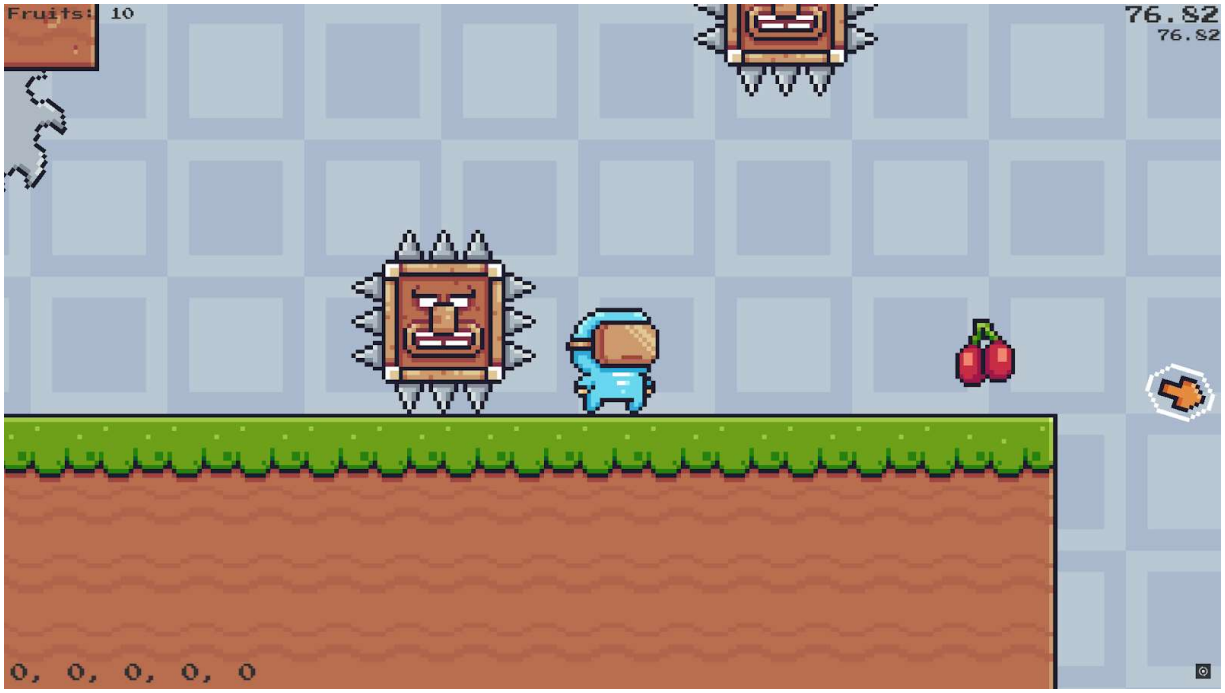
Slime killed when jumped on



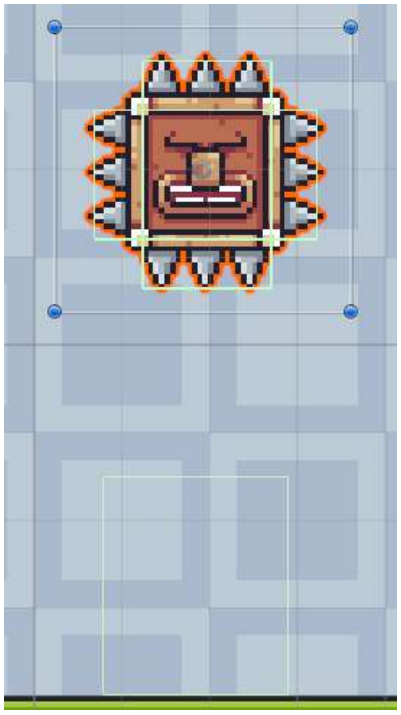
Can adjust animation time depending on specific enemy's animation time

Falling Spike Head

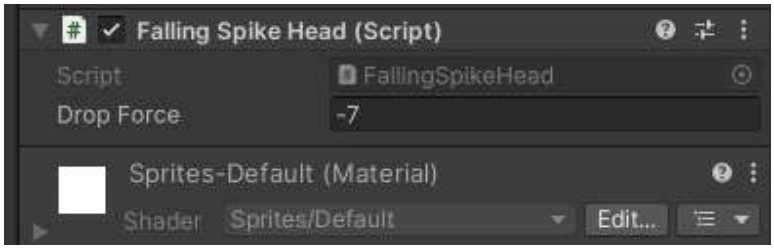
```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FallingSpikeHead : MonoBehaviour
{

    //Rigidbody2D variable so spike head can be moved
    private Rigidbody2D rb;
    //Drop force can be changed from Unity editor
    [SerializeField] private float dropForce = -10f;

    private void Start()
    {
        //Gets Rigidbody2D component on start
        rb = GetComponent<Rigidbody2D>();
    }

    //Trigger box is under spike head (on ground)
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //Checks that the object in the trigger box is the player
        if (collision.gameObject.name == "Player")
        {
            //Drops spike head
            rb.velocity = new Vector2(rb.velocity.x, dropForce);
        }

    }

}
```



Spike head falls when player walks into trigger box below it and will kill the player if they collide



Falling spike head and trigger box

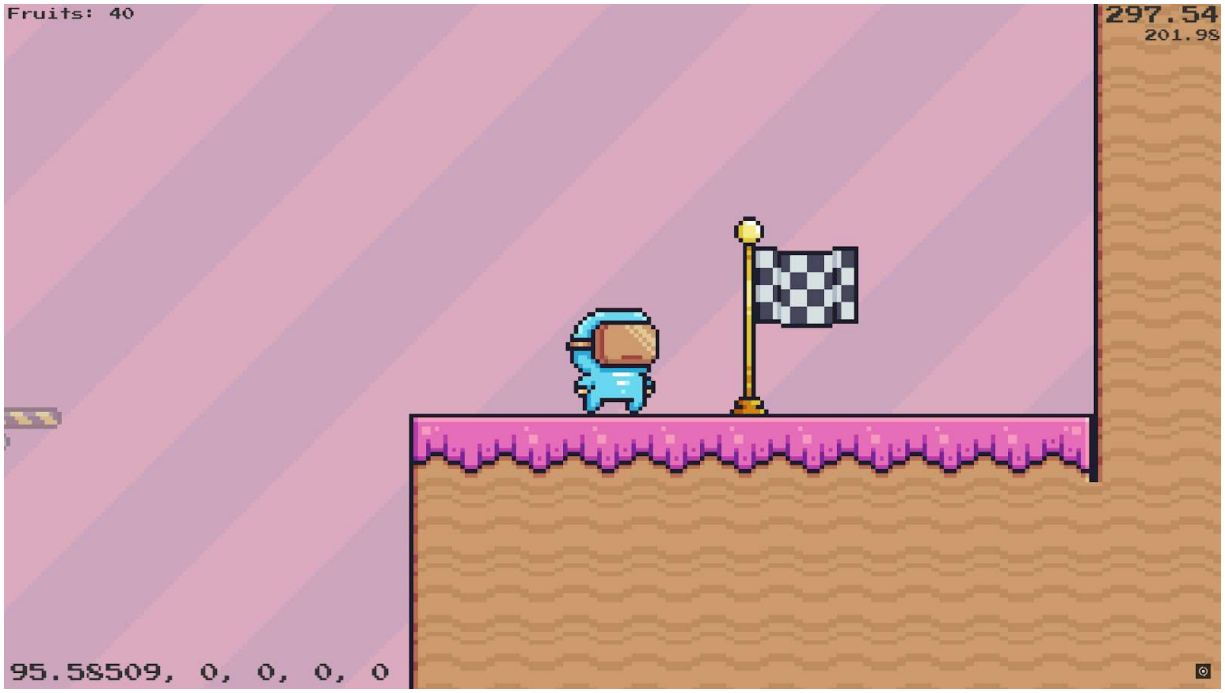


Can change drop force in the editor

Finish

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports SceneManagement from the Unity engine to allow us to manage scenes
using UnityEngine.SceneManagement;
public class Finish : MonoBehaviour
{
    //Sets finish sound effect variable
    private AudioSource finishSound;
    //Creates boolean variable to dictate if level is complete
    public static bool levelCompleted = false;

    private CollectablesManager collectablesManager;

    public void Awake()
    {
        collectablesManager =
GameObject.FindGameObjectsWithTag("Collectables")[0].GetComponent<CollectablesM
anager>();
        GameObject timer = GameObject.FindGameObjectsWithTag("Timer")[0];
        timer.GetComponent<Timer>().finishScript = this;
    }

    // Start is called before the first frame update
    private void Start()
    {
        //Gets audio source for finish sound
        finishSound = GetComponent<AudioSource>();
    }

    //OnTriggerEnter2D is from Unity library
```

```
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //If player touches flag trigger and level is completed
        if (collision.gameObject.name == "Player" && !levelCompleted)
        {
            //Plays finish sound
            finishSound.Play();
            //Makes level completed boolean true
            levelCompleted = true;
            //Calls complete level after 2f seconds
            Invoke("CompleteLevel", 2f);
        }
    }


    //Function/void to call when level is completed
    private void CompleteLevel()
    {
        levelCompleted = false;
        collectablesManager.SaveFruits();
        //Loads the level of the next build index
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

}
```



Finish flag

Loads next level after 2f seconds

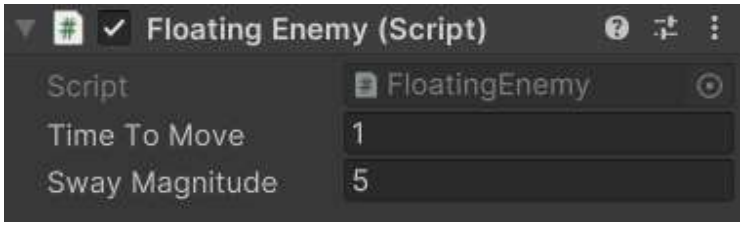
Final finish line

Floating Enemy

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FloatingEnemy : MonoBehaviour
{
    //Values to change how the flyuing object moves
    [SerializeField] private float timeToMove = 1f;
    [SerializeField] private float swayMagnitude = 3f;
    //Gets Rigidbody2D of the player
    private Rigidbody2D rb;
    //Boolean is true if the game object has an increasing y value
    private bool movingUp = true;

    void Start()
    {
        //Gets Rigidbody2D of the game object
        rb = GetComponent<Rigidbody2D>();
```

```
        //Starts the swaying
        StartCoroutine(Sway());
    }

    private IEnumerator Sway()
    {
        //Always continues during the game
        while (true)
        {
            //Alternates between moving up and moving down
            if (movingUp)
            {
                rb.velocity = new Vector2(rb.velocity.x, swayMagnitude);
            }
            else
            {
                rb.velocity = new Vector2(rb.velocity.x, -swayMagnitude);
            }

            movingUp = !movingUp;

            //Waits for specified time before next movement
            yield return new WaitForSeconds(timeToMove);
        }
    }
}
```

Bat changes height and flies between waypoints



Can adjust time to move and sway magnitude in the editor

Flying Item

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FlyingItem : MonoBehaviour
{
    //Gets animator
    private Animator anim;
    //Gets Rigidbody2D
    private Rigidbody2D rb;
    //X and Y direction values
```

```
private float dirX = 0f;
private float dirY = 0f;
//Movement speed can be changed in Unity editor
[SerializeField] private float moveSpeed = 5f;
//Boolean for whether flying item is active
private bool active;

private void Start()
{
    //Gets components on start frame
    anim = GetComponent<Animator>();
    anim.SetTrigger("Idle");
    rb = GetComponent<Rigidbody2D>();
    dirX = Input.GetAxisRaw("Horizontal");
    dirY = Input.GetAxisRaw("Vertical");
    rb.velocity = new Vector2(dirX * moveSpeed, rb.velocity.y);
}

private void OnTriggerStay2D(Collider2D collision)
{
    //On collision activates animation
    anim.SetTrigger("Active");
    //Sets active boolean to true
    active = true;
}

private void OnTriggerExit2D(Collider2D collision)
{
    //On end of collision deactivates animation *
    anim.SetTrigger("Idle");
    //Sets active boolean to false
    active = false;
}

private void Update()
{
    //Calls movement subroutine every frame if flying item is active
    if (active)
    {
        Movement();
    }
}

//Allows X and Y movement of flying item
```

```
    //Not affected by gravity so can move freely
    private void Movement()
    {
        //Takes X and Y input
        dirX = Input.GetAxisRaw("Horizontal");
        dirY = Input.GetAxisRaw("Vertical");
        //Apply input to vector, proportional to moveSpeed
        rb.velocity = new Vector2(dirX * moveSpeed, dirY * moveSpeed);
    }

}
```

Flying item carries player and takes inputs when player is inside it

Can adjust movement speed in the editor

Import API

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Allows JSON parsing
using UnityEngine.Networking;
//Allows text to be changed
using UnityEngine.UI;

public class ImportAPI : MonoBehaviour
{
    //Text for quote and author to be written to added in Unity editor
    [SerializeField] private Text quoteText;
    //API URL to be used
    private const string API_URL = "https://zenquotes.io/api/random";
    //Array of default quotes to be used if API request fails
    private string[] defaultQuotes = {"Have a great day!", "Well played!",
"Play again soon!"};

    void Start()
    {
        StartCoroutine(FetchQuote());
    }

    private IEnumerator FetchQuote()
    {
        using (UnityWebRequest webRequest = UnityWebRequest.Get(API_URL))
        {
            //Requests quote
            yield return webRequest.SendWebRequest();

            //If API request fails, a random quote from defaultQuotes array is
used instead
            if (webRequest.result == UnityWebRequest.Result.ConnectionError ||
webRequest.result == UnityWebRequest.Result.ProtocolError)
            {
                quoteText.text = defaultQuotes[Random.Range(0,
defaultQuotes.Length)];
            }
            else
            {
                //JSON parsing
```

```
                //Gets response
                string jsonResponse = webRequest.downloadHandler.text;
                //Gets quote part
                string quote = jsonResponse.Split(new[] { "\"q\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
                //Gets author part
                string author = jsonResponse.Split(new[] { "\"a\":\"" },
System.StringSplitOptions.None)[1].Split('\"')[0];
                //Concatenates author and quote with space and outputs to the
quoteText
                quoteText.text = quote + " - " + author;
            }
        }
    }
}
```

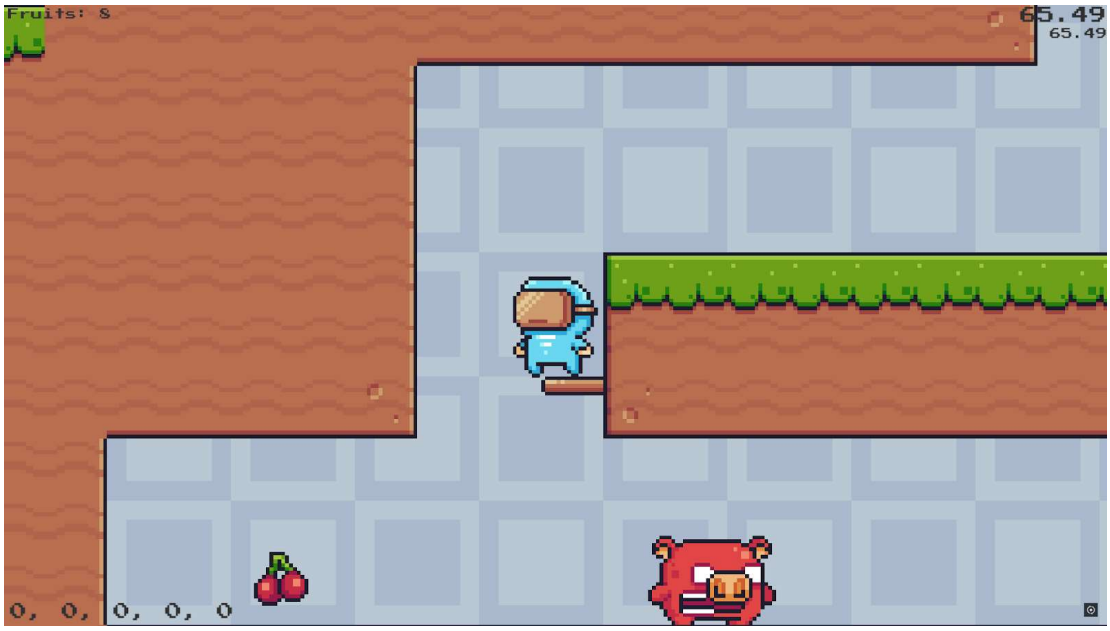
End screen with API quote (online)

End screen with a random default quote (offline)

Item Collector

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Also imports Unity UI
using UnityEngine.UI;

public class ItemCollector : MonoBehaviour
{
    //Sets fruit counter to 0
    private int fruits = 0;
    //Allows fruit text overlay to be inserted in Unity editor
    public Text FruitsText;
    //Allows collection sound effect to be inserted in Unity editor
    [SerializeField] private AudioSource collectionSoundEffect;

    private CollectablesManager collectablesManager;
```

```
    public void Awake()
    {
        collectablesManager =
GameObject.FindGameObjectsWithTag("Collectables")[0].GetComponent<CollectablesM
anager>();
    }

    //Imports OnTriggerEnter2D
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //Runs on collision with fruit
        if (collision.gameObject.CompareTag("Fruits"))
        {
            //Destroys fruit
            Destroy(collision.gameObject);

            collectablesManager.AddFruits(1);

            //Plays fruit collection sound effect
            collectionSoundEffect.Play();
        }
    }

}
```



Fruit counter increments by 1 each time a fruit is collected

On/Off Platform

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OnOffPlatform : MonoBehaviour
{
    //Float to determine how long the game object stays in its current state
before changing
    [SerializeField] private float timePerState = 1f;
    //Float to initially delay the platform changing state
    [SerializeField] private float initialDelay = 0f;

    //Creates sprite variable
    private SpriteRenderer sprite;
    //Creates box collier 2D variable
    private BoxCollider2D collider;

    // Start is called before the first frame update
    void Start()
    {
        //Sets sprite and boxcollider 2D variables to the components of the
game object
        sprite = GetComponent<SpriteRenderer>();
        collider = GetComponent<BoxCollider2D>();
        //Start the pltform toggle subroutine
        StartCoroutine(TogglePlatform());
    }


    private IEnumerator TogglePlatform()
    {
        yield return new WaitForSeconds(initialDelay);
        while (true)
        {
            //Waits for time of the active state
            yield return new WaitForSeconds(timePerState);
            //Dims platform colour
            sprite.color = new Color(100, 100, 100, .5f);
            //Disables box collider 2D of game object
            collider.enabled = false;
            //Waits for time of the active state
```

```
            yield return new WaitForSeconds(timePerState);
            //Returns platform to normal colour
            sprite.color = Color.white;
            //Re-enables the box collider 2D
            collider.enabled = true;
        }

    }
}
```

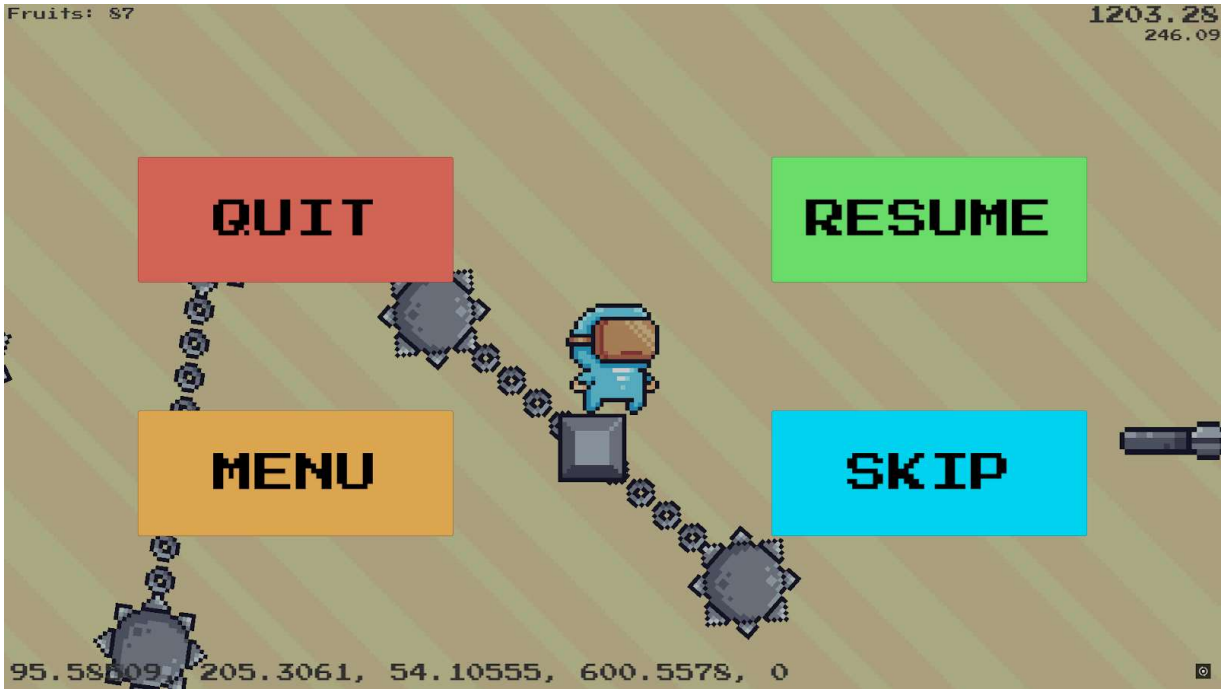
Platforms alternate between off (left/right) and on (middle)

Values can be adjusted in the editor to change time in each state and delay to offset alternating platforms from each other

Pause Menu

```C/C++
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;

public class PauseMenu : MonoBehaviour
{
    //Booleans for if the game is paused or the level is skipped
    public static bool gameIsPaused = false;
    public static bool levelSkipped = false;
    //Reference to pause menu UI
    public GameObject pauseMenuUI;

    void Update()
    {
        //Toggles pause menu using escape key
        if (Input.GetKeyDown(KeyCode.Escape) ||
Input.GetKeyDown(KeyCode.JoystickButton7))
        {
            if (gameIsPaused)
            {
                Resume();
            }

            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        //Reverts levelSkipped to false in case this function has been called
from the skip level function
        levelSkipped = false;
        //Disables UI
        pauseMenuUI.SetActive(false);
        //Returns time scale to normal so the game can be played
        Time.timeScale = 1f;
        //Bool indicates game is no longer paused
        gameIsPaused = false;
    }
```

```
    void Pause()
    {
        //Inverse of resume
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        gameIsPaused = true;
    }

    public void LoadMenu()
    {
        //Normal time scale set
        Time.timeScale = 1f;
        //Returns to start menu scene
        SceneManager.LoadScene(0);
        //Unpause the game
        Resume();
    }

    public void QuitGame()
    {
        //Quits game completely
        Application.Quit();
    }

    public void SkipLevel()
    {
        //Bool shows level has been skipped
        levelSkipped = true;
        //Loads the level of the next build index
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        //Resumes level with Resume() function
        Resume();
    }
}
```



Pause menu with buttons to quit game, resume game, return to start screen/menu, skip level

Persistent UI

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PersistentUI : MonoBehaviour
{
    private void Awake()
    {
        //Gets an array of persistent UI game objects
        GameObject[] objs = GameObject.FindGameObjectsWithTag("PersistentUI");

        //Deletes any new ones that have been created
        if (objs.Length > 1)
        {
            Destroy(gameObject);
        }

        //Ensures the UI is carried over to the next scene
```

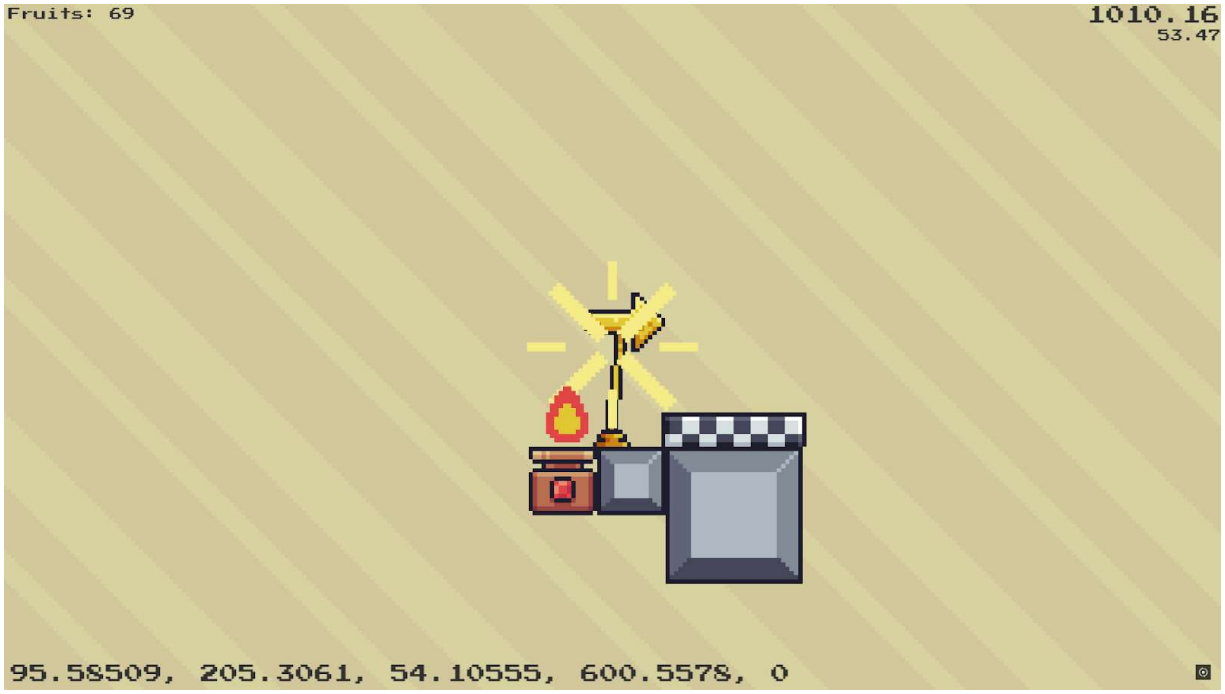
```
            DontDestroyOnLoad(gameObject);
        }
    }
```

UI continues throughout levels, fruits and global times increment throughout levels, local time is just for each level and times array is always for levels 1-5 respectively

Player Life

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports Unity scene management
using UnityEngine.SceneManagement;

public class PlayerLife : MonoBehaviour
{
    //Gets animator and rigidbody 2D for player
    private Animator anim;
    private Rigidbody2D rb;

    //Allows death sound effect to be inserted in Unity editor
    [SerializeField] private AudioSource deathSoundEffect;

    private void Start()
    {
        //Initiates animator and rigidbody 2D on start
        anim = GetComponent<Animator>();
        rb = GetComponent<Rigidbody2D>();
    }
    //OnCollision2D method
    private void OnCollisionEnter2D(Collision2D collision)
    {
        //Detects trap
        if (collision.gameObject.CompareTag("Trap"))
        {
            //Runs death void/function
            Die();
        }
    }

    //Void/function runs when player dies
    public void Die()
    {
        //Stops movement of player
        rb.bodyType = RigidbodyType2D.Static;
        //Triggers death animation
        anim.SetTrigger("death");
        //Plays death sound effect
        deathSoundEffect.Play();
    }
```

```
    private void RestartLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```



Death

Player Movement

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    //Gets rigidbody component so it does not have to be called every frame
(private so only this script)
    private Rigidbody2D rb;
    //Stores character Animator component as variable anim
    private Animator anim;
    //DirX can now be called in seperate void to main update void, default
value of 0f so is not undefined later
    private float dirX = 0f;
    //Assigns sprite renderer in engine to variable sprite
    private SpriteRenderer sprite;
    //Gives movement speed a variable
    //[SerializeField] allows value to be changed within the Unity editor
    //To reset to script values right click script in Unity editor and reset
script
    //Public to be changed when mushroom item collected
    [SerializeField] public float playerMoveSpeed = 7f;
    //Gives jump force/height a variable
    [SerializeField] private float jumpForce = 14f;
    //Enum allows variable to hold own own set values, each has corresponding
int value (hold mouse over it to see)
    private enum MovementState {Idle, Running, Jumping, Falling}
    //Gets BoxCollider2D of player
    private BoxCollider2D coll;
    //Calls layer of terrain used to check if player is on the ground
    [SerializeField] private LayerMask jumpableGround;
    //Allows jump sound file to be inserted in the editor
    [SerializeField] private AudioSource jumpSoundEffect;

    // Start is called before the first frame update
    private void Start()
    {
        //Gets all variables on first frame / start
        rb = GetComponent< Rigidbody2D>();
        anim = GetComponent<Animator>();
        sprite = GetComponent<SpriteRenderer>();
```

```
        coll = GetComponent<BoxCollider2D>();
    }

    // Update is called once per frame
    private void Update()
    {
        //Horizontal Movement, includes joystick support, raw to stop character
sliding after moving
        dirX = Input.GetAxisRaw("Horizontal");
        rb.velocity = new Vector2(dirX * playerMoveSpeed, rb.velocity.y);



        // Jump movement (Currently space bar change to contoller later (Unity
input system))
        // && is "and" for if statement
        if (Input.GetButtonDown("Jump") && IsGrounded())
        {
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            jumpSoundEffect.Play();
        }

        UpdateAnimationState();

    }

    private void UpdateAnimationState()
    {
        MovementState state;

        //Running animation
        //If move forward running animation true
        if (dirX > 0f)
        {
            state = MovementState.Running;
            sprite.flipX = false;
        }
        //If move backwards running animation true and sprite flips to face
left/backwards
        else if (dirX < 0f)
        {
            state = MovementState.Running;
            sprite.flipX = true;
        }
```

```
        //If not moving then idle animation plays instead
        else
        {
            state = MovementState.Idle;
        }

        //Sets animation when jumping (increasing height)
        //Checks for grounding in case on platform moving up
        if (rb.velocity.y > 0.1f && IsGrounded() == false)
        {
            state = MovementState.Jumping;
        }
        //Sets animation when falling (decreasing height)
        //Checks for grounding in case on platform moving down
        else if (rb.velocity.y < -0.1f && IsGrounded() == false)
        {
            state = MovementState.Falling;
        }

        //Sets animation state after loops which dictate animation state (using
int value of our enum)
        anim.SetInteger("state", (int)state);
    }

    private bool IsGrounded()
    {
        //Box colliders
        //Creates box cast around player collider box to check if player is on
the ground/ near walls
        //Used so player can only jump on ground, not above ground or by walls
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f,
Vector2.down, 0.1f, jumpableGround);
    }
}
```

Idle

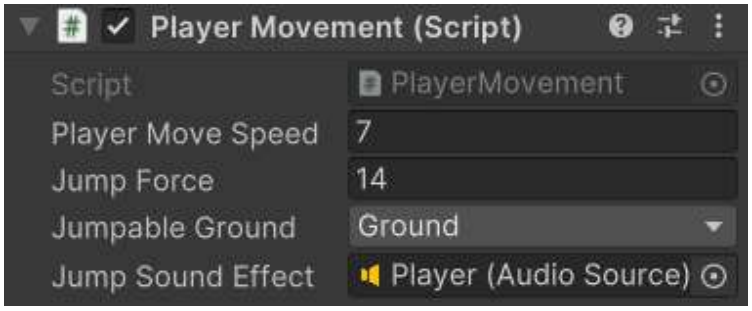
Running


Fruits: 69
977.70
21.09
95.58509, 205.3061,

Jump


Fruits: 69
981.56
24.94
95.58509, 205.3061,

Fall

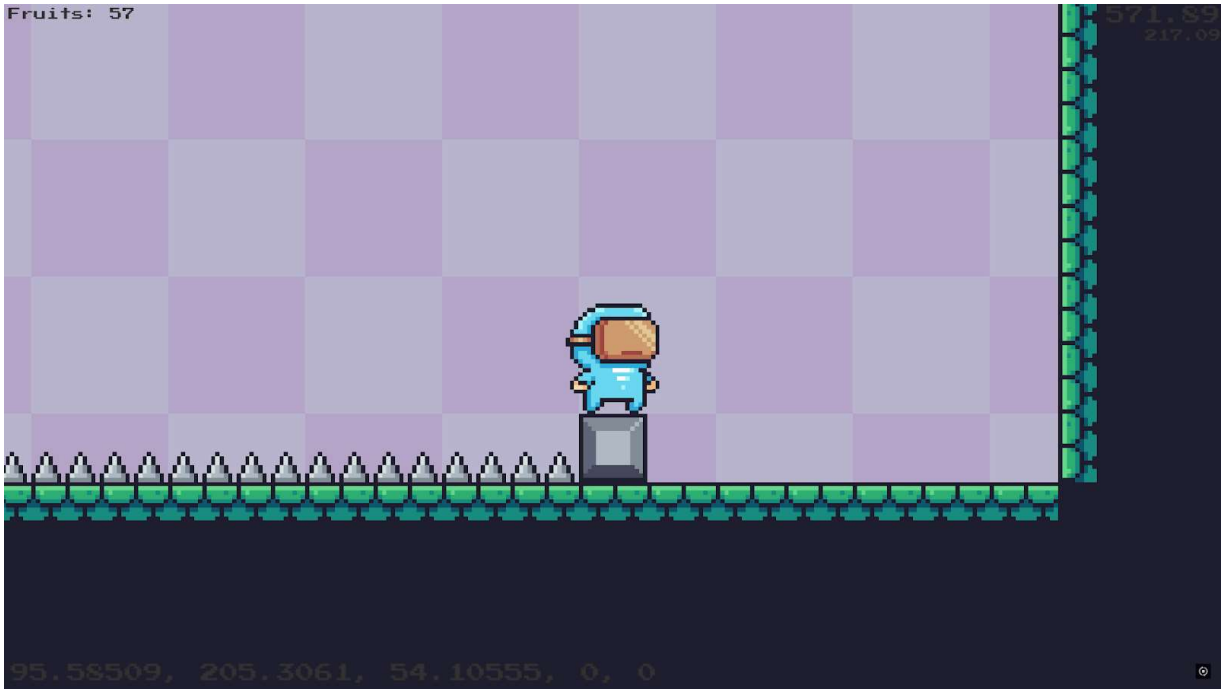


Can adjust movement speed and jump force in the editor

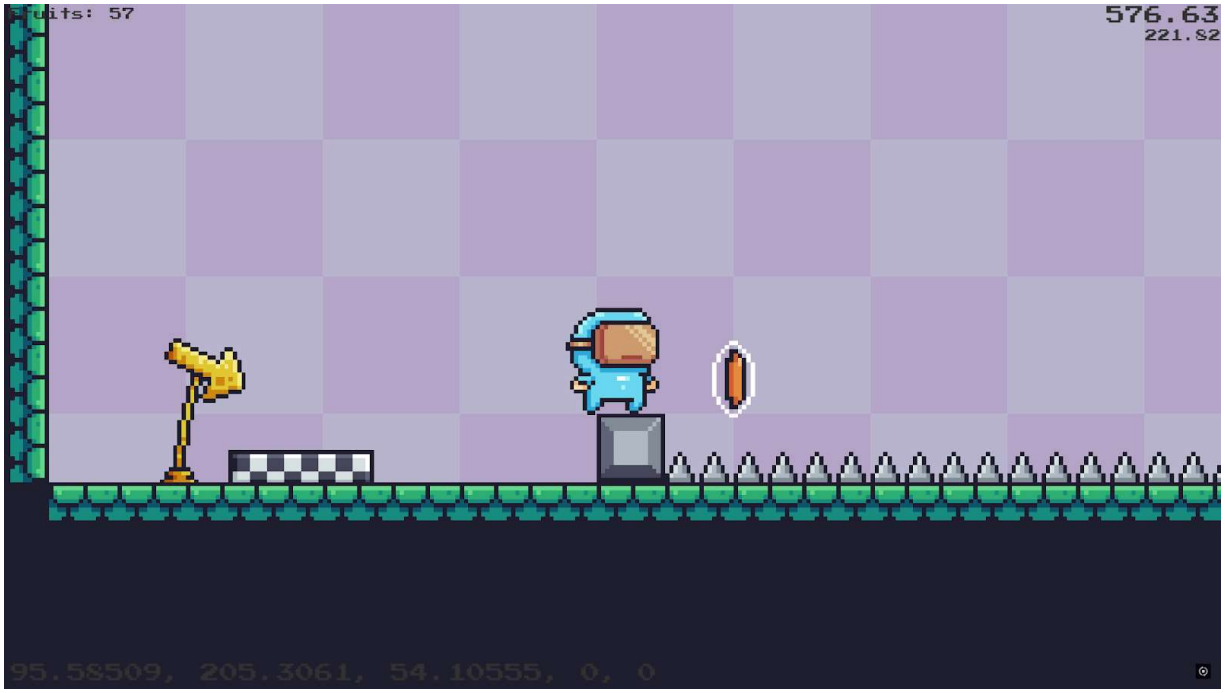
Recall

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Recall : MonoBehaviour
{
    //Stacks for corresponding X and Y values of recall points
    private Stack<float> stackX = new Stack<float>();
    private Stack<float> stackY = new Stack<float>();
    //Box collider for checking if the player is on the ground
    private BoxCollider2D coll;
    //Allows ground to be the layer to be checked
    [SerializeField] private LayerMask ground;

    //Allows Rigidbody2D of player to be input in the editor
    [SerializeField] private Rigidbody2D rb;
    //Inserts audio sources for recall and setting recall in the editor
    [SerializeField] private AudioSource spawnSet;
    [SerializeField] private AudioSource recall;

    private void Start()
    {
        //Gets box collider of player
        coll = GetComponent<BoxCollider2D>();
    }

    void Update()
    {
        //Recall point only triggered if the player is on the ground
```

```
        if (Input.GetButtonDown("SetSpawn") && IsGrounded())
        {
            //Plays recall point set sound effect
            spawnSet.Play();
            //Pushes X and Y coordinates of recall position onto corresponding
stacks
            stackX.Push(transform.position.x);
            stackY.Push(transform.position.y);
        }

        //Recall only occurs if the X and Y stacks are empty
        if (Input.GetButtonDown("Recall") && stackX.Count != 0 && stackY.Count
!= 0)
        {
            //Plays recall sound effect
            recall.Play();
            //Pops X and Y values of player from corresponding stacks and
changes player position to these
            rb.MovePosition(new Vector3(stackX.Pop(), stackY.Pop(),
transform.position.z));
        }
    }

    private bool IsGrounded()
    {
        //Box colliders
        //Creates box cast around player collider box to check if player is on
the ground/ near walls
        //Used so player can only jump on ground, not above ground or by walls
        return Physics2D.BoxCast(coll.bounds.center, coll.bounds.size, 0f,
Vector2.down, 0.1f, ground);
    }
}
```



Recall triggered

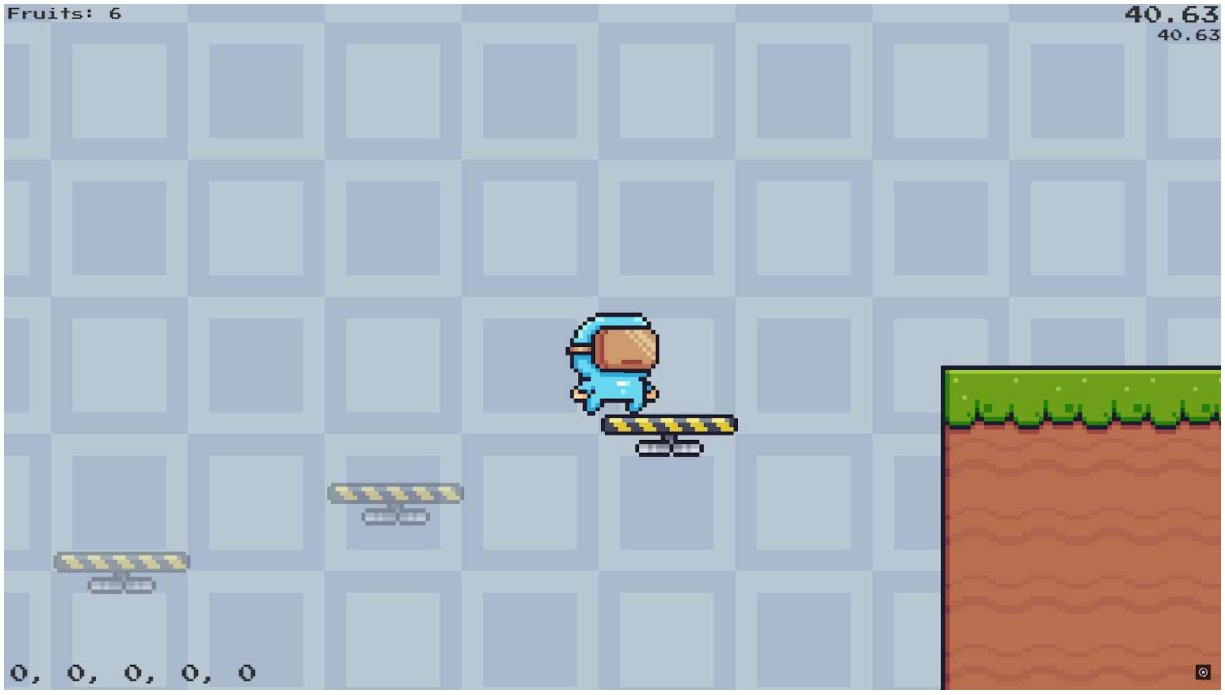


Recalled to old position

Recurring Platform

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RecurringPlatform : MonoBehaviour
{
    //Allows time to destroy to be changed from Unity engine
    [SerializeField] private float timeToDestroy = 1f;
    [SerializeField] private float timeToRespawn = 1f;

    //Sprite and box collider 2D variables
    private SpriteRenderer sprite;
    private BoxCollider2D collider;

    private void Start()
    {
        //Gets sprite renderer and box collider 2D components of recurring
platform
        sprite = GetComponent<SpriteRenderer>();
        collider = GetComponent<BoxCollider2D>();
    }

    //Detects collision
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.name == "Player")
        {
            //Calls subroutine DisappearAfterDelay()
            StartCoroutine(DisappearAfterDelay());
        }
    }

    private IEnumerator DisappearAfterDelay()
    {
        //Waits for 1 second
        yield return new WaitForSeconds(timeToDestroy);
        //Disables the platform
        //Dims platform sprite
        sprite.color = new Color(100, 100, 100, .5f);
        //Disables platform collider
        collider.enabled = false;
```

```
            //Waits for respawn for timeToRespawn seconds
            yield return new WaitForSeconds(timeToRespawn);
            //Reverts colour to normal
            sprite.color = Color.white;
            //Re-enables collider
            collider.enabled = true;
        }
    }
```

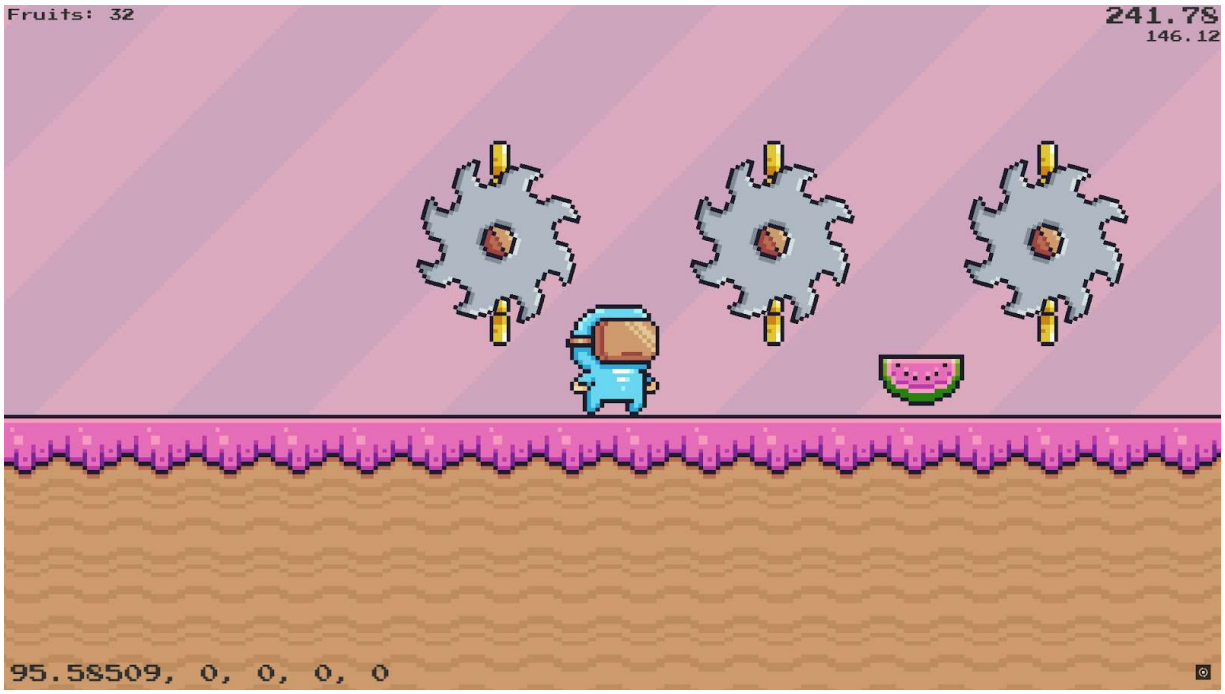


Platforms disable (left) x seconds after first being stood on and return to active (right) y seconds later

Can adjust time to destroy and time to respawn in the editor

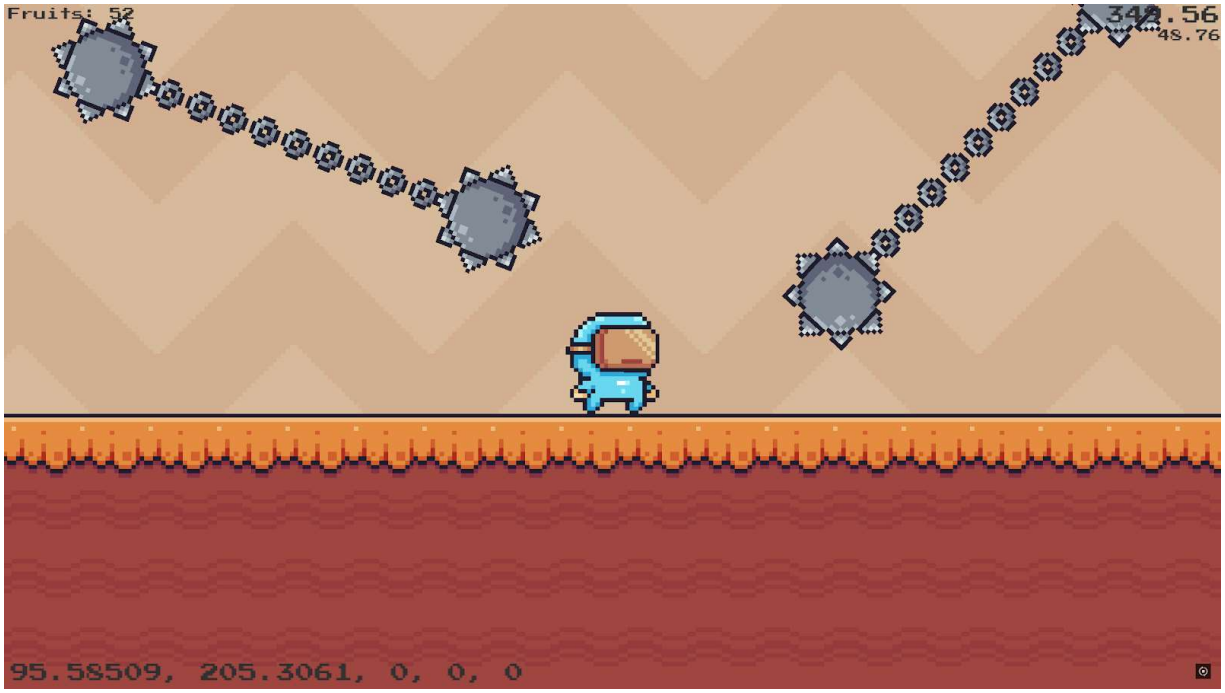
Rotate

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rotate : MonoBehaviour
{
    //Rotate speed set in editor
    [SerializeField] private float speed = 2f;


    private void Update()
    {
        //Rotates game object at speed set in editor
        //Not affected by frame rate as it is multiplied by Time.deltaTime
        transform.Rotate(0, 0, 360 * speed * Time.deltaTime);
    }
}
```

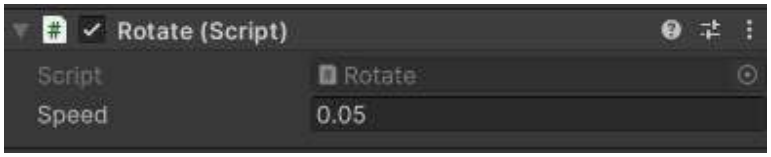


Saws rotate

Rotating platforms rotate



Spike balls rotate

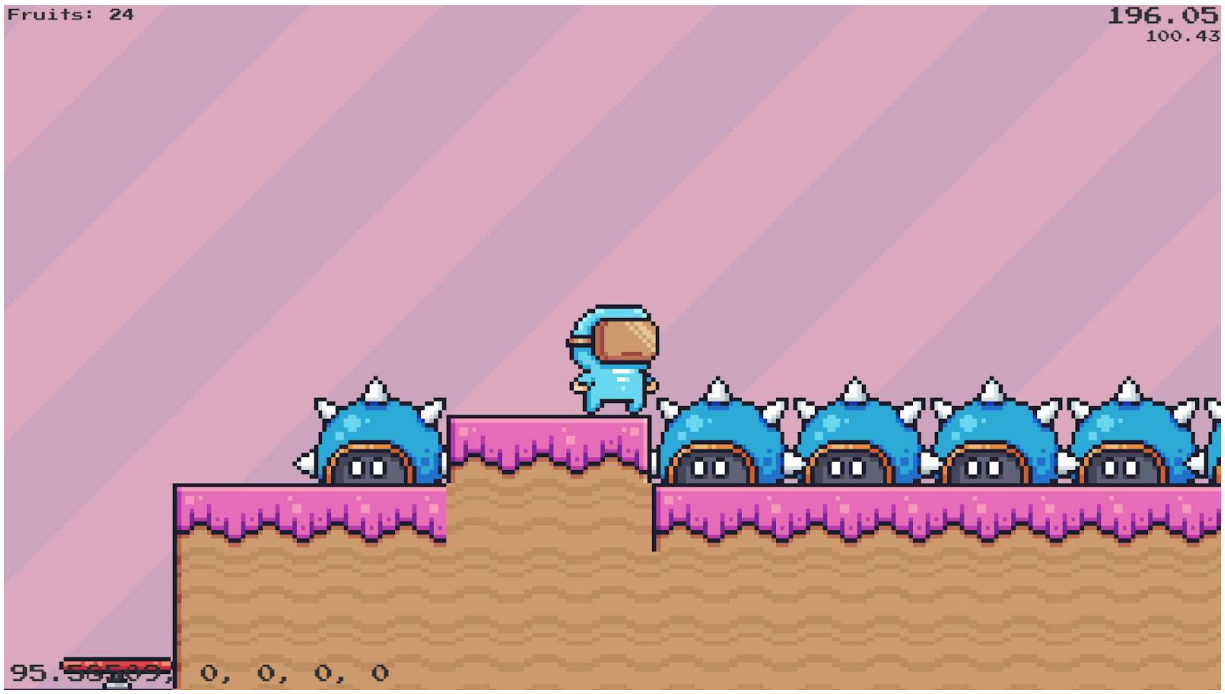


Can adjust speed of rotation in the editor

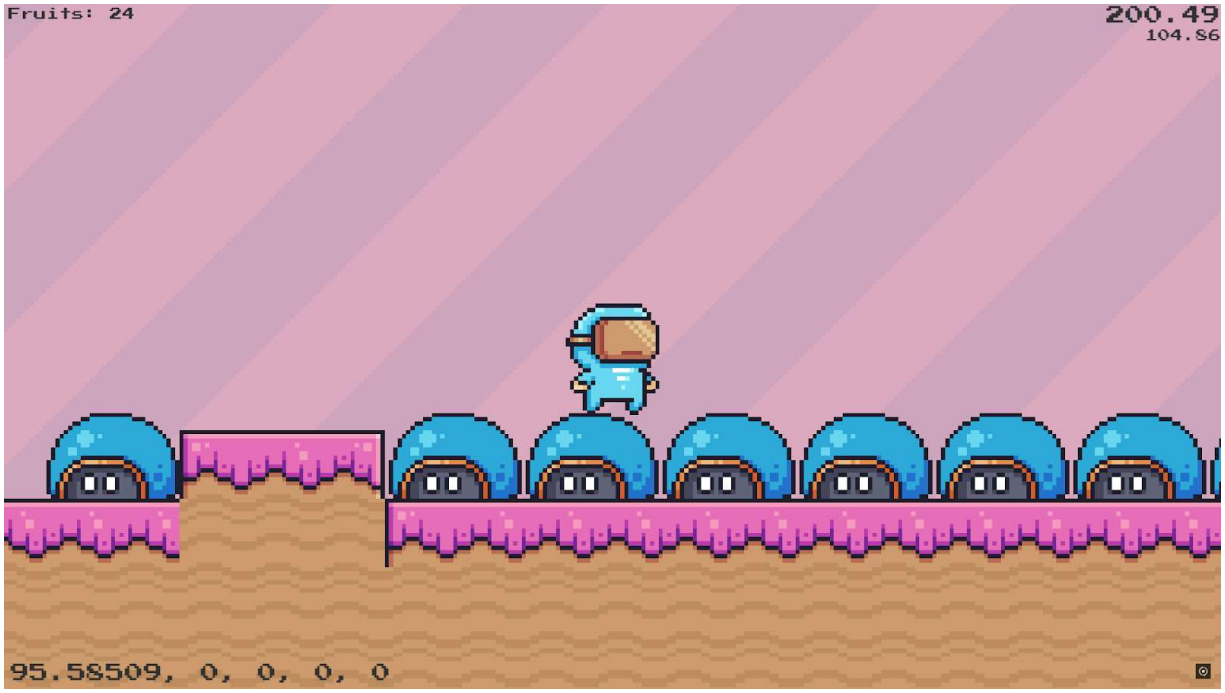
Spike Turtle

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpikeTurtle : MonoBehaviour
{
    private Animator anim;
    //Boolean value showing active state
    private bool active = true;
    //5.05 seconds is time turtle is active before deactivating
    private float timeToWait = 5.05f;

    //Start is called before the first frame update
    void Start()
    {
        anim = GetComponent<Animator>();
        //Start the coroutine to toggle the active state
        StartCoroutine(ToggleState());
    }

    private IEnumerator ToggleState()
    {
        while (true)
        {
            //Wait for the end of current state (5.05 seconds)
            yield return new WaitForSeconds(timeToWait);
            //Toggle the active state with subroutine
            ToggleActive();
        }
    }

    private void ToggleActive()
    {
        //Invert boolean
        active = !active;
        //Change animation state
        anim.SetBool("active", active);

        // Change the tag and layer based on the new state
        if (active)
        {
```

```
            //Trap so can kill player
            gameObject.tag = "Trap";
            //Default layer as no need to be jumpable
            gameObject.layer = LayerMask.NameToLayer("Default");
        }
        else
        {
            //Untagged so cannot hurt player
            gameObject.tag = "Untagged";
            //Ground so shell can be jumped on
            gameObject.layer = LayerMask.NameToLayer("Ground");
        }
    }
}
```



Spike turtles active

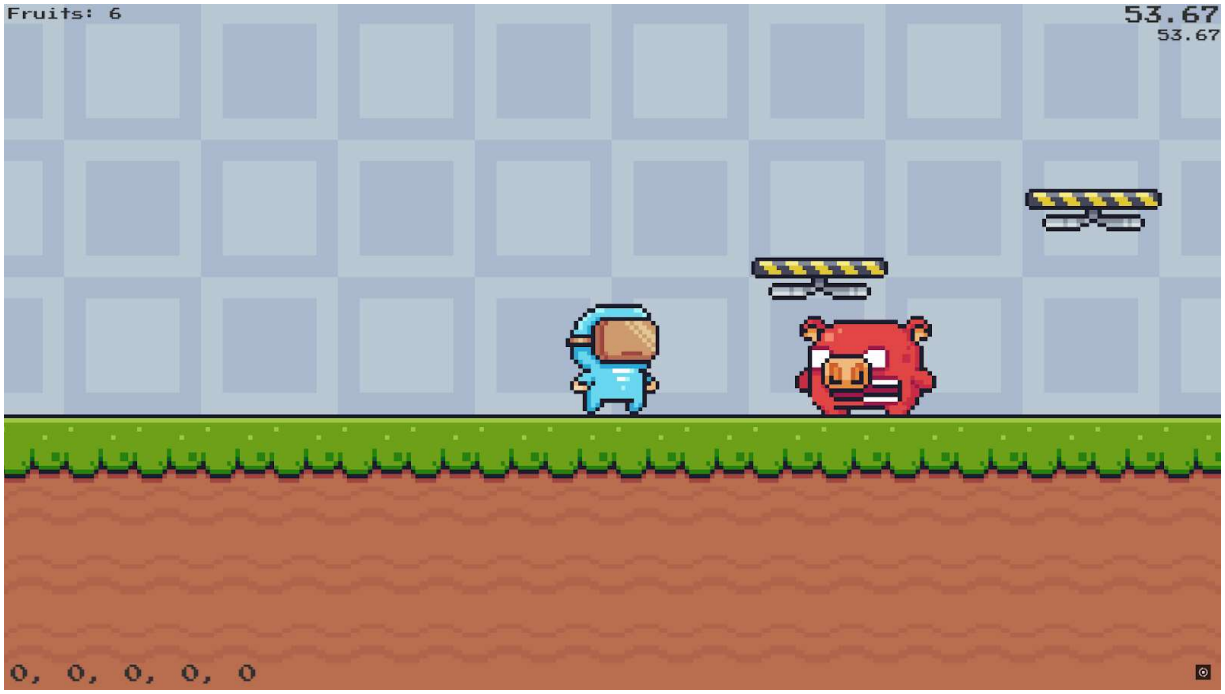


Spike turtles not active so can be walked on and jumped on

Sprite Flip Left

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpriteFlipLeft : MonoBehaviour
{
    // Reference to the WaypointFollower script
    [SerializeField] private WaypointFollower waypointFollower;

    // Update is called once per frame
    private void Update()
    {
        // Ensure waypointFollower is assigned
        if (waypointFollower == null)
        {
            Debug.LogError("WaypointFollower script is not assigned.");
            return;
        }
```

```
        // Get the current waypoint the object is moving towards
        Vector3 targetPosition =
waypointFollower.Waypoints[waypointFollower.CurrentWaypointIndex].transform.pos
ition;
        // Determine the direction to the target
        Vector3 direction = targetPosition - transform.position;

        // Check if the object is moving left or right
        if (direction.x > 0)
        {
            // If moving right, flip the sprite to face left
            transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (direction.x < 0)
        {
            // If moving left, flip the sprite to face right
            transform.localScale = new Vector3(1, 1, 1);
        }
    }
}
```



Flips sprite such as angry pig so it always faces in the direction its moving in

Sprite Flip Right

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SpriteFlipRight : MonoBehaviour
{
    // Reference to the WaypointFollower script
    [SerializeField] private WaypointFollower waypointFollower;

    // Update is called once per frame
    private void Update()
    {
        // Ensure waypointFollower is assigned
        if (waypointFollower == null)
        {
            Debug.LogError("WaypointFollower script is not assigned.");
            return;
        }

        // Get the current waypoint the object is moving towards
        Vector3 targetPosition =
waypointFollower.Waypoints[waypointFollower.CurrentWaypointIndex].transform.pos
ition;
        // Determine the direction to the target
        Vector3 direction = targetPosition - transform.position;

        // Check if the object is moving left or right
        if (direction.x < 0)
        {
            // If moving left, flip the sprite to face left
            transform.localScale = new Vector3(-1, 1, 1);
        }
        else if (direction.x > 0)
        {
            // If moving right, flip the sprite to face right
            transform.localScale = new Vector3(1, 1, 1);
        }
    }
}
```

Start Menu

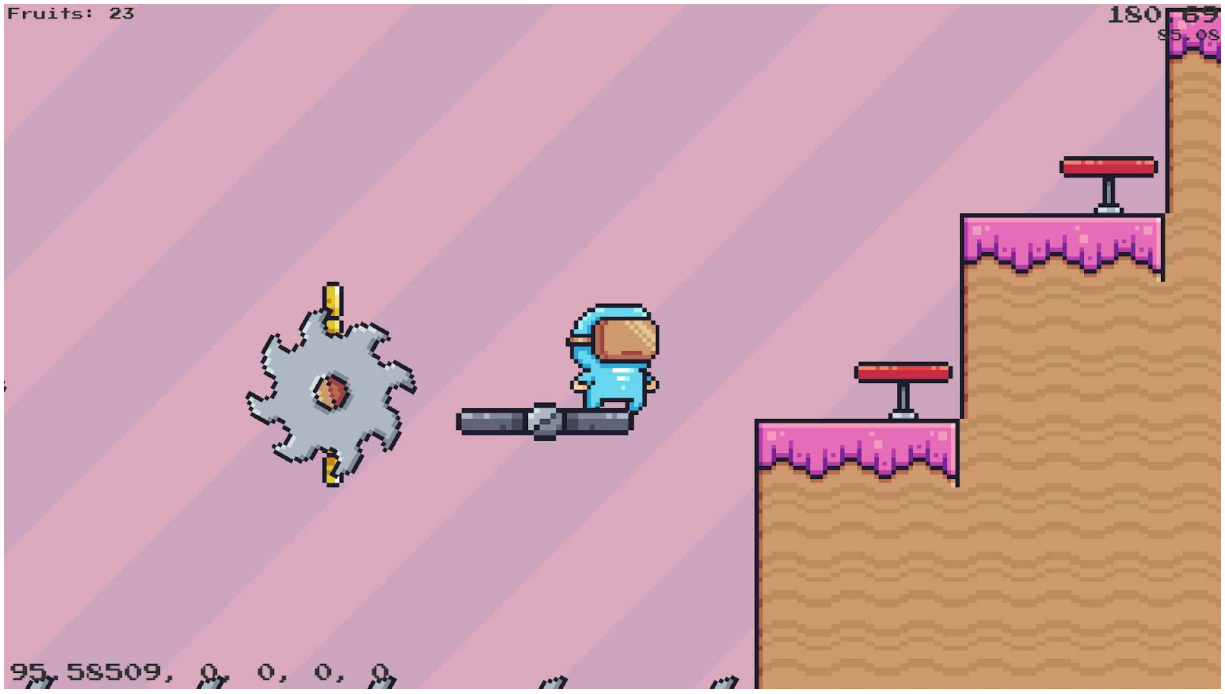
```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports Unity scene manager
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    //Public so it can be seen in Unity engine to be added to on click () part
of button component
    public void StartGame()
    {
        //Loads the next scene (level 1)
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }

    //Quits Game
    public void Quit()
    {
        Application.Quit();
    }
}
```

Start menu with buttons to start (load level 1) or quit the game entirely

Sticky Platform

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StickyPlatform : MonoBehaviour
{
    //Both use "Trigger" which is only for top box collider of moving platform

    //OnTriggerEnter2D part of Unity Library
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //if collided with player
        if (collision.gameObject.name == "Player")
        {
            //Sets player game object as a child of moving platform
            collision.gameObject.transform.SetParent(transform);
        }
    }
```

```
    //OnTriggerExit2D part of Unity library
    private void OnTriggerExit2D(Collider2D collision)
    {
        //Sets player game object as not a child of moving platform
        collision.gameObject.transform.SetParent(null);
    }
}
```



Moving platform carries player with it and player movement is now relative to the movement of the moving platform

Swing

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Swing : MonoBehaviour
{
    //Rate of rotation in degrees per second
    [SerializeField] private float degreesPerSecond = 180f;
    //Wait time in seconds
    [SerializeField] private float waitTime = 1f;
    //Bool to dictate clockwise start
    [SerializeField] private bool clockwise = true;
    //Scale to be applied to rotation to make it start clockwise/anticlockwise
    private float directionScale = 1f;
    //Float of time to wait before swinging starts
    [SerializeField] private float initialDelay = 0f;

    void Start()
    {
        //Positive rotation for clockwise
        if (clockwise)
        {
            directionScale = 1f;
        }
        //Negative rotation for anticlockwise
        else
        {
            directionScale = -1f;
        }
        //Starts rotating on first frame
        StartCoroutine(Rotate());
    }

    IEnumerator Rotate()
    {
        //Waits for initial delay so swings can be offset
        yield return new WaitForSeconds(initialDelay);
        //Always rotate while game is active
        while (true)
        {
            //Rotate 180 degrees clockwise in z axis
```

```
            yield return StartCoroutine(RotateByAngle(Vector3.forward, 180f *
directionScale));


            //Waits for specified time
            yield return new WaitForSeconds(waitTime);

            //Rotate 180 degrees anticlockwise in z axis
            yield return StartCoroutine(RotateByAngle(Vector3.forward, -180f *
directionScale));

            //Waits for specified time
            yield return new WaitForSeconds(waitTime);
        }
    }

    IEnumerator RotateByAngle(Vector3 axis, float angle)
    {
        //Targets position to specified angle
        float targetAngle = Mathf.Abs(angle);
        //Total angle rotated starts from 0
        float rotatedAngle = 0f;

        //Loop until the total rotated angle reaches the target angle
        while (rotatedAngle < targetAngle)
        {
            //Calculate rotation step based off speed and change in time so not
affected by frame rate
            float step = degreesPerSecond * Time.deltaTime;

            //Checks if step exceeds target angle
            if (rotatedAngle + step > targetAngle)
            {
                //Avoids swing over-rotating
                step = targetAngle - rotatedAngle;
            }
            //Rotates angle by calculated step
            transform.Rotate(axis, Mathf.Sign(angle) * step);
            //Adds step to total angle rotated
            rotatedAngle += step;
            //Returns nothing
            yield return null;
        }
    }
```

```
}
```

Platform swings

Can also swing in the other direction

Can adjust swing speed, time stationary between swings, offset swing patterns and change swing direction in editor

Timer

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Imports Unity UI
using UnityEngine.UI;
//Imports SceneManagement from the Unity engine to allow us to check if game compelete screen is reached
using UnityEngine.SceneManagement;
using System;
//Allows writing to text files
using System.IO;

public class Timer : MonoBehaviour
{
    //Allow timer texts to be added in Unity editor
    //Total game time is public becuase it is continued throughout levels
    public Text timerText;
    [SerializeField] private Text localTimeText;
    [SerializeField] private Text arrayText;

    //Float values for time
    private float currentTime = 0f;
    private float localTime = 0f;

    //Bools to show if the level has been completed or skipped
    private bool TimeEnd = false;
    private bool skipped = false;
```

```
    //References to finish script and pause menu script
    public Finish finishScript;
    public PauseMenu pauseMenu;

    //Array for times to complete each level
    private float[] times = new float[5];
    //Array of times to be sorted by merge sort
    private float [] sortedTimes = new float[5];
    //Bool to show if time is saved to times array
    private bool written = false;
    //Bool to show if text file has been written to
    private bool fileWritten = false;

    private void Update()
    {


        //Writes times to Timer text file and resets timer when the last level
is completed
        if ((SceneManager.GetActiveScene().buildIndex) ==
(SceneManager.sceneCountInBuildSettings - 1))
        {
            SaveTimesToFile();

            //Resets local and global timers to 0f in case the game is
restarted
            currentTime = 0f;
            localTime = 0f;
        }
        else
        {
            //Resets fileWritten bool when game is restarted from end screen
            fileWritten = false;
            //Gets boolean values for the finish being triggered or the level
being skipped
            TimeEnd = Finish.levelCompleted;
            skipped = PauseMenu.levelSkipped;

            //Only increments if level is not complete and level has not been
skipped
            if (!TimeEnd && !skipped)
            {
                IncrementTime();
                written = false;
```

```
            }
            else
            {
                TimeToArray();
                //Resets local timer to 0.00 aftfer level is completed/skipped
                localTime = 0f;
                localTimeText.text = localTime.ToString("0.00");
            }

        }

    }

    private void SaveTimesToFile()
    {
        //fileWritten bool ensures file is only appended to once per game
completion
        if (!fileWritten)
        {
            //Adds line
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**************************************************************************
**********************" + Environment.NewLine);
            //Appends todays date
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
DateTime.Today.ToShortDateString() + Environment.NewLine);

            //Appends each level number and time by iterating with a for loop
            for (int i = 0; i < 5; i++)
            {
                File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"Level " + (i + 1).ToString() + ":   " + times[i].ToString() +
Environment.NewLine);
            }

            //Appends total (global) time
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Total
time:   " + currentTime.ToString() + Environment.NewLine);
            //Appends array of times 1-5 respectively
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times
for levels 1-5 respectively:   " + string.Join(", ", times) +
Environment.NewLine);
            //Performs a merge sort on the times array
            sortedTimes = MergeSort(times);
```

```
            //Appends times for levels 1-5 in order from shortest to longest
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt", "Times
sorted by shortest to longest:   " + string.Join(", ", sortedTimes) +
Environment.NewLine);
            //Adds line
            File.AppendAllText("C:\\Users\\james\\Desktop\\Times.txt",
"**********************************************************************
**********" + Environment.NewLine);
            //Ensures file is not appended to again
            fileWritten = true;
        }
    }
    private void IncrementTime()
    {
        //Increments global timer with change in time since last frame
        currentTime += Time.deltaTime;
        //Updates global timer text to 2 decimal places
        timerText.text = currentTime.ToString("0.00");

        //Increments local timer with change in time since last frame
        localTime += Time.deltaTime;
        //Updates local timer text to 2 decimal places
        localTimeText.text = localTime.ToString("0.00");
    }

    private void TimeToArray()
    {
        if (!written)
        {
            //Saves local time for level to times array
            int levelIndex = SceneManager.GetActiveScene().buildIndex - 1;

            if (levelIndex >= 0 && levelIndex < times.Length)
            {
                times[levelIndex] = localTime;
                arrayText.text = string.Join(", ", times);
                written = true;
            }
        }
    }


    //Merge sort function
    private static float[] MergeSort(float[] array)
```

```
    {
        //Checks if array is just 1 element long or empty
        if (array.Length <= 1)
        {
            return array;
        }

        //Calculates mid point
        int midPoint = array.Length / 2;

        //Creates left and right half arrays
        float[] left = new float[midPoint];
        float[] right = new float[array.Length - midPoint];

        //Populates left and right arrays with respective contents from array
        Array.Copy(array, 0, left, 0, midPoint);
        Array.Copy(array, midPoint, right, 0, array.Length - midPoint);

        //Recursively sort left and right halves until the array is completely
sorted
        left = MergeSort(left);
        right = MergeSort(right);

        //Sort left and right halves using Merge function and return sorted
array
        return Merge(left, right);
    }

    //Merges two arrays
    private static float[] Merge(float[] left, float[] right)
    {
        float[] result = new float[left.Length + right.Length];

        //Position in left array
        int i = 0;
        //Position in right array
        int j = 0;
        //Position in "array" array
        int k = 0;

        //Compare elements from the left and right arrays and merge them in
sorted order
        while (i < left.Length && j < right.Length)
        {
```

```
            //Comparison
            if (left[i] <= right[j])
            {
                //If left array value is smaller, it is written to "array"
array
                result[k++] = left[i++];
            }
            else
            {
                //If right array value is smaller, it is written to "array"
array
                result[k++] = right[j++];
            }
        }

        //Copy any remaining elements from the left array to "array"
        while (i < left.Length)
        {
            result[k++] = left[i++];
        }

        //Copy any remaining elements from the right array to "array"
        while (j < right.Length)
        {
            result[k++] = right[j++];
        }

        //Returns 1 sorted array
        return result;
    }

    //Keeps track of when a new scene is loaded
    private void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }

    //Keeps track of when a level ends
    private void OnDisable()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }
```

```
    //Triggers ResetTimesArray(), local and global times at the start of level
1 in case game is restarted
    private void OnSceneLoaded(Scene scene, LoadSceneMode mode)
    {
        if (scene.buildIndex == 1)
        {
            ResetTimesArray();
            localTime = 0f;
            currentTime = 0f;
        }
    }

    //Resets times array to 0, 0, 0, 0, 0
    private void ResetTimesArray()
    {
        //Iterates through the loop, reseting each element to 0f
        for (int i = 0; i < times.Length; i++)
        {
            times[i] = 0f;
        }
        arrayText.text = string.Join(", ", times);
    }


}
```

Base case of the merge sort is when a list has 0 or 1 elements. Time complexity of merge sort is O(n log n) so it is very efficient.

Global and local times (top right) and times array for levels 1-5 respectively bottom left

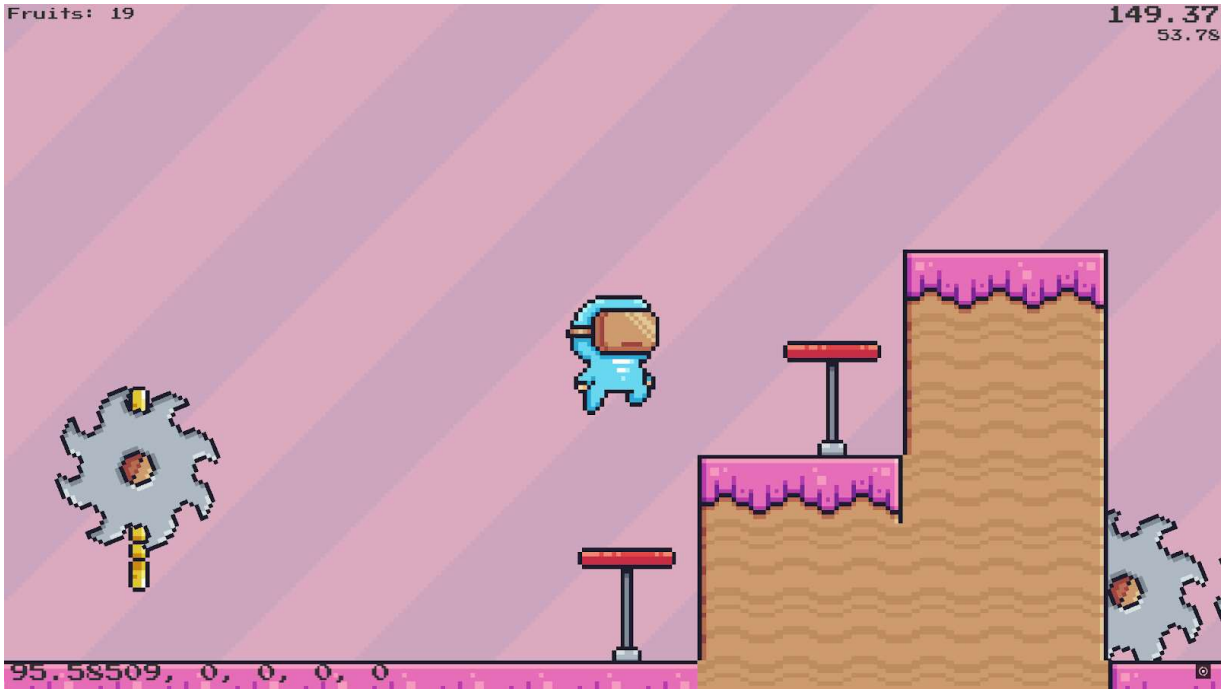
```
**************************************************************************************************
17/01/2025
Level 1:   95.58509
Level 2:   205.3061
Level 3:   54.10555
Level 4:   600.5578
Level 5:   337.2529
Total time:   1294.814
Times for levels 1-5 respectively:   95.58509, 205.3061, 54.10555, 600.5578, 337.2529
Times sorted by shortest to longest:   54.10555, 95.58509, 205.3061, 337.2529, 600.5578
**************************************************************************************************
```

Text file output

Trampoline

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Trampoline : MonoBehaviour
{
    //Player Rigidbody2D assigned in Unity editor
    [SerializeField] private Rigidbody2D rb;
    //Jump force can be changed in Unity editor
```

```
    [SerializeField] private float jumpForce = 14f;
    //Jump sound audio source input in Unity editor
    [SerializeField] private AudioSource jumpSoundEffect;

    //Trigger box on top of trampoline
    private void OnTriggerEnter2D(Collider2D collision)
    {
        //Checks that game object in trigger box is the player
        if (collision.gameObject.name == "Player")
        {
            //Applies jump vector to Rigidbody2D of player
            rb.velocity = new Vector2(rb.velocity.x, jumpForce);
            //Plays jump sound effect
            jumpSoundEffect.Play();
        }
    }

}
```



Trampoline launches player up

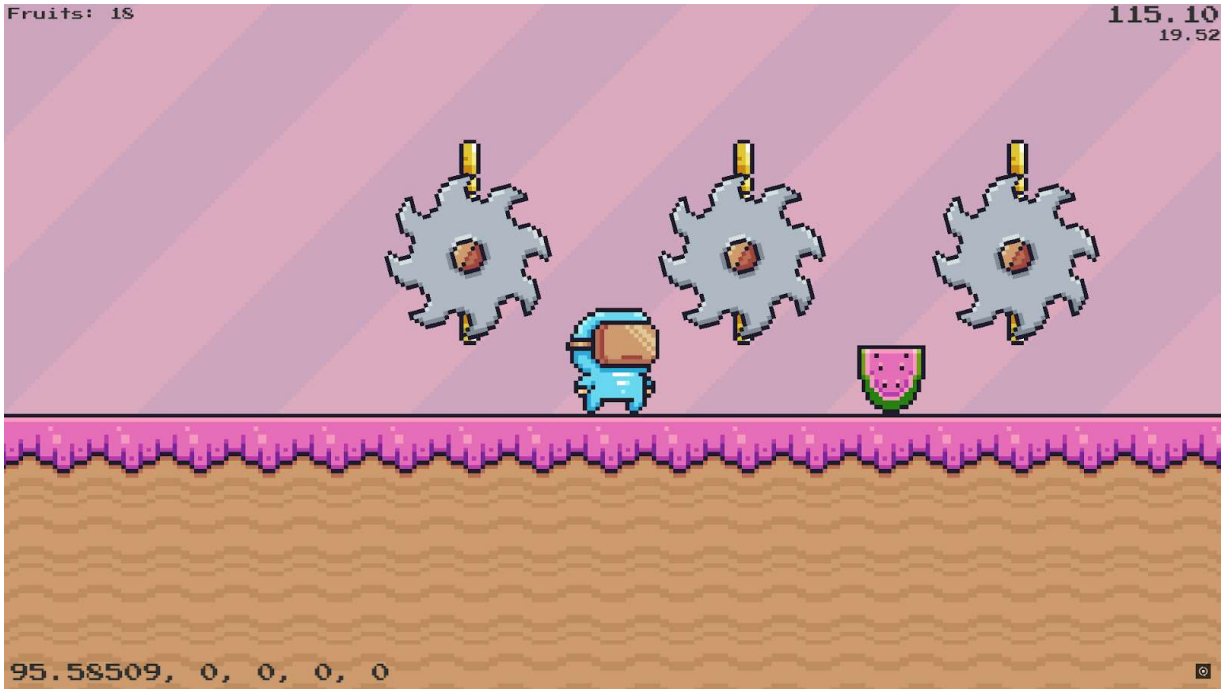
Fan launches player up

Jump force can be changed in the editor

Waypoint Follower

```C/C++
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class WaypointFollower : MonoBehaviour
{
    //Creates a list of waypoints to follow, serialized field so can be changed
in the Unity editor
    [SerializeField] private GameObject[] waypoints;
```

```
    //Int variable for current waypoint index being moved towards from the list
of waypoints
    private int currentWaypointIndex = 0;

    //Float value for speed of moving game object
    [SerializeField] private float speed = 2f;

    //Public because they need to be accessed in the editor
    //Current waypoint game object
    public GameObject[] Waypoints => waypoints;
    //Current waypoint index (corresponds to list of waypoints)
    public int CurrentWaypointIndex => currentWaypointIndex;

    private void Update()
    {
        //Checks distance between moving game object and waypoint
        if
(Vector2.Distance(waypoints[currentWaypointIndex].transform.position,
transform.position) < 0.1f)
        {
            //Increments currentWaypointIndex by 1 when very close to waypoint
            currentWaypointIndex++;
            //Reverts index to 0 if index exceeds size of list
            if (currentWaypointIndex >= waypoints.Length)
            {
                currentWaypointIndex = 0;
            }
        }
        // Moves the game object each frame
        // Time.deltaTime makes speed move from clock not frame rate
        transform.position = Vector2.MoveTowards(transform.position,
waypoints[currentWaypointIndex].transform.position, Time.deltaTime * speed);
    }
}
```



Traps move via waypoints

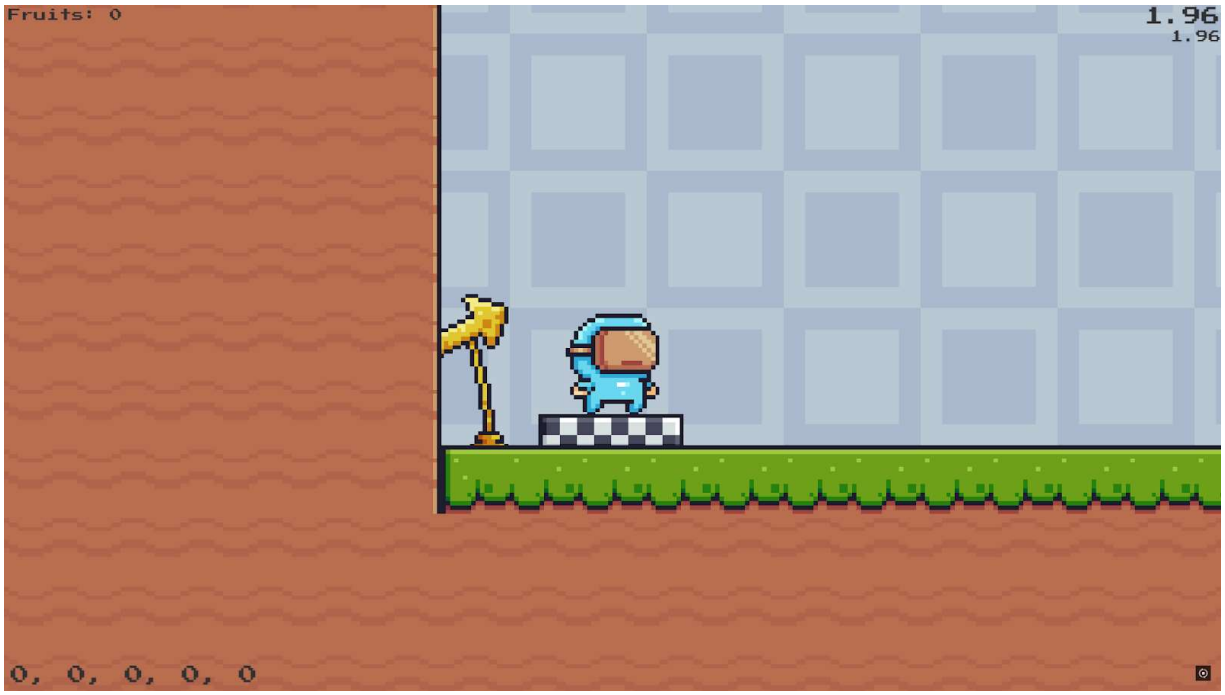
Moving platforms move via waypoints

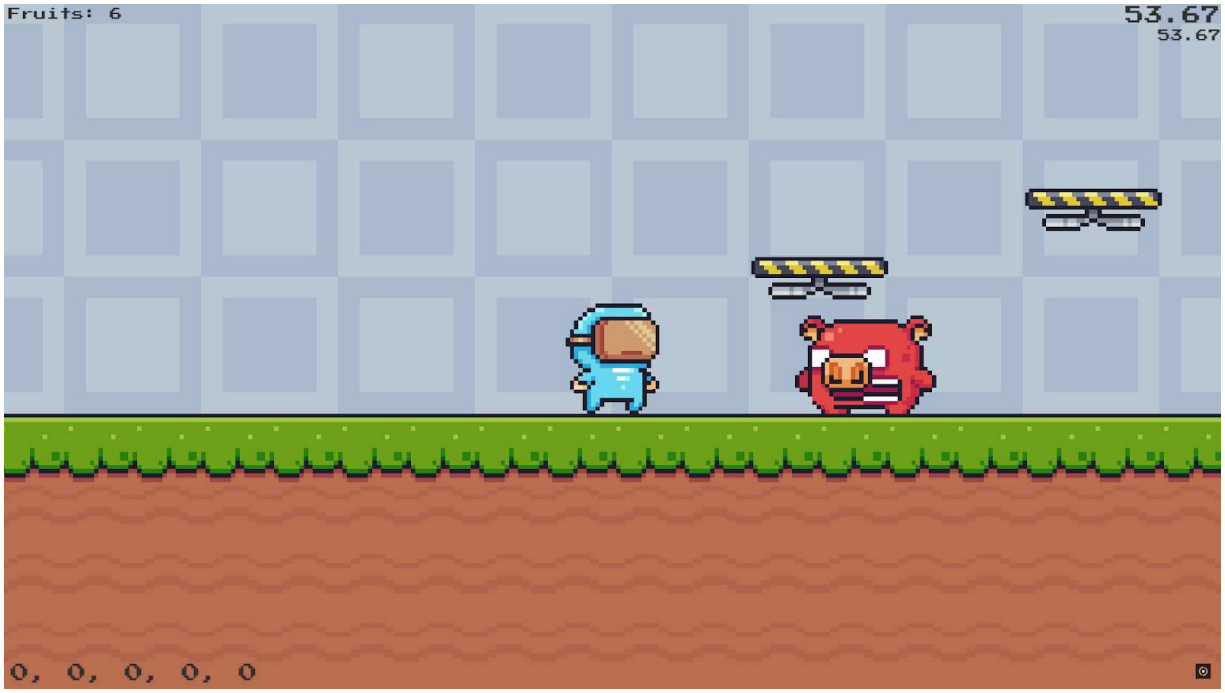
Platforms follow end chains as waypoints

Waypoint game objects are added to the dynamic list in the editor.

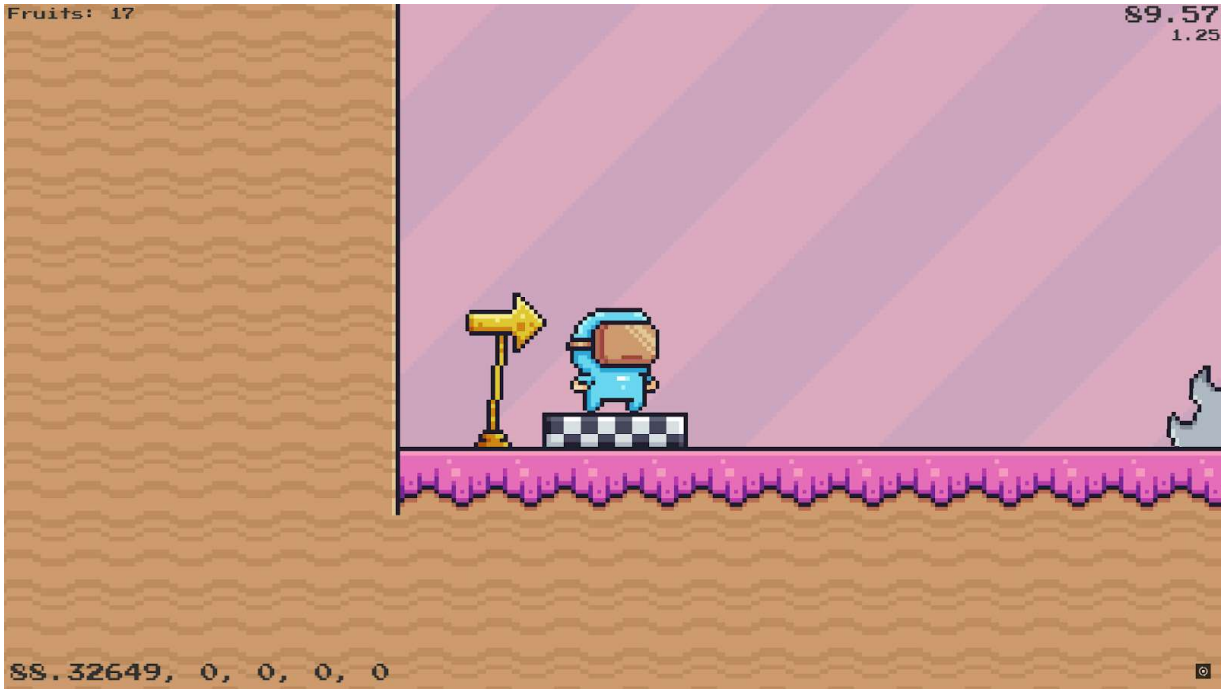
Gameplay and text file screenshots

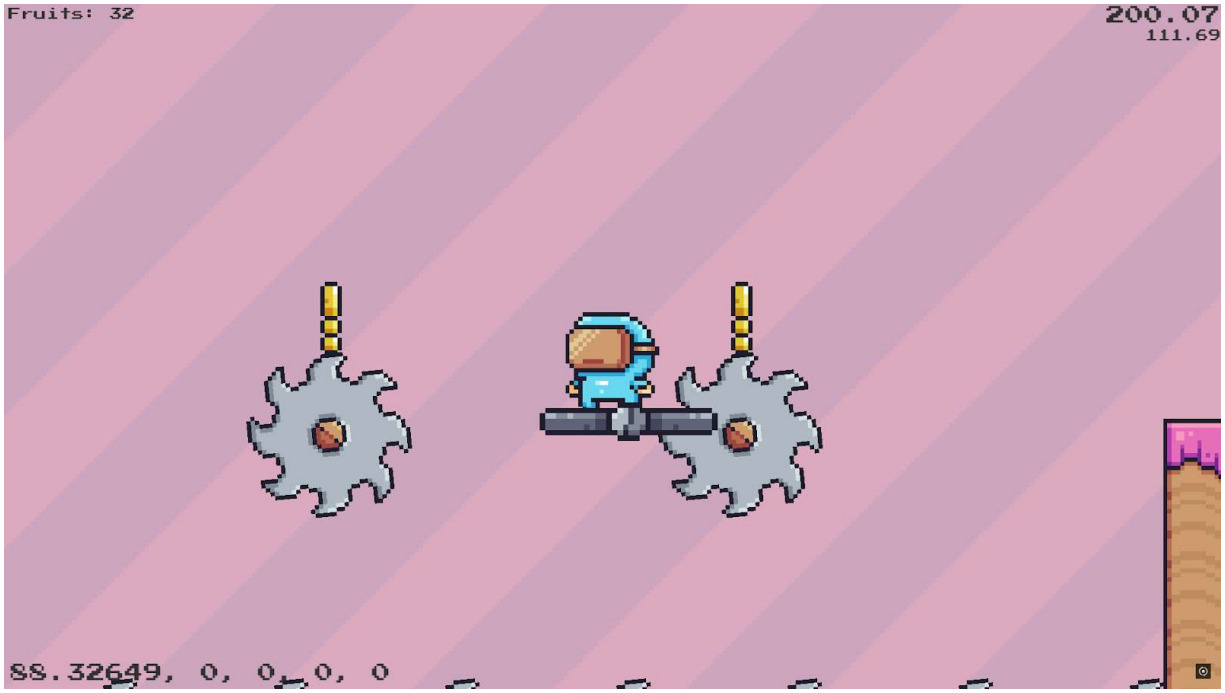
Start screen





Level 1




Fruits: 32
200.07
111.69
88.32649, 0, 0, 0, 0

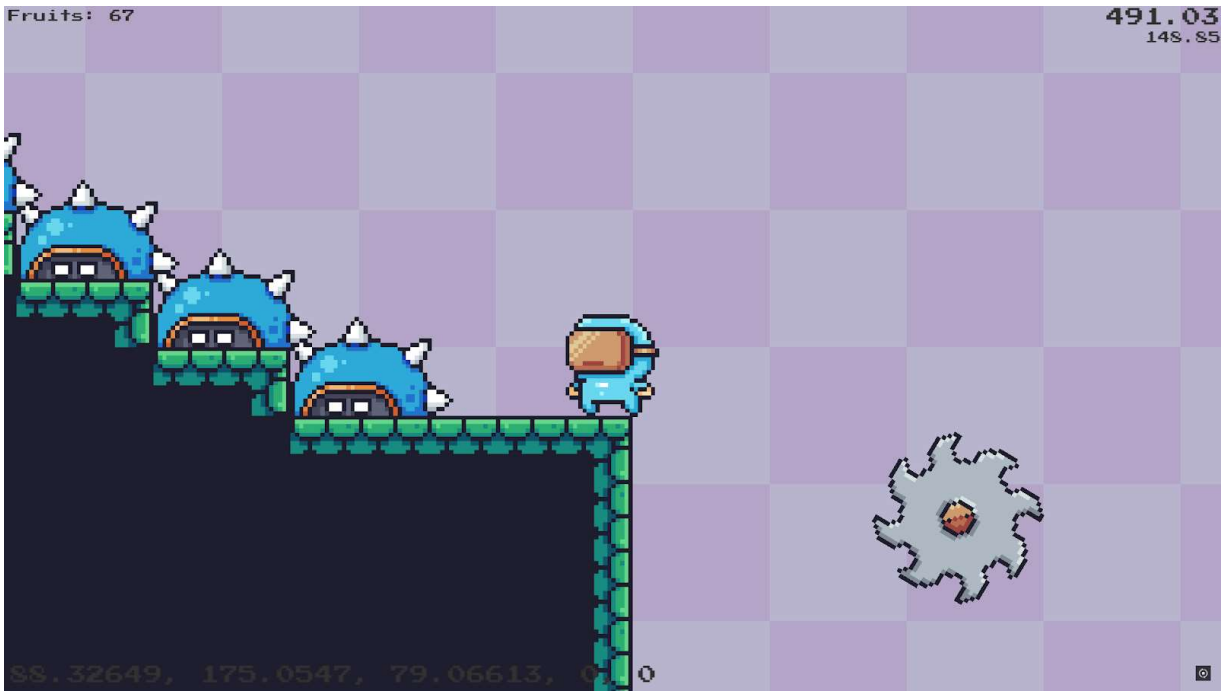
Level 2


Fruits: 43
264.60
1.16
88.32649, 175.0547, 0, 0, 0


Fruits: 62
88.32649, 175.0547, 0, 0, 0

Level 3


348.09
88.32649, 175.0547, 79.06613, 0, 0


Fruits: 67
491.03
148.85
88.32649, 175.0547, 79.06613, 0, 0


Fruits: 72
584.25
241.96
88.32649, 175.0547, 79.06613, 0, 0

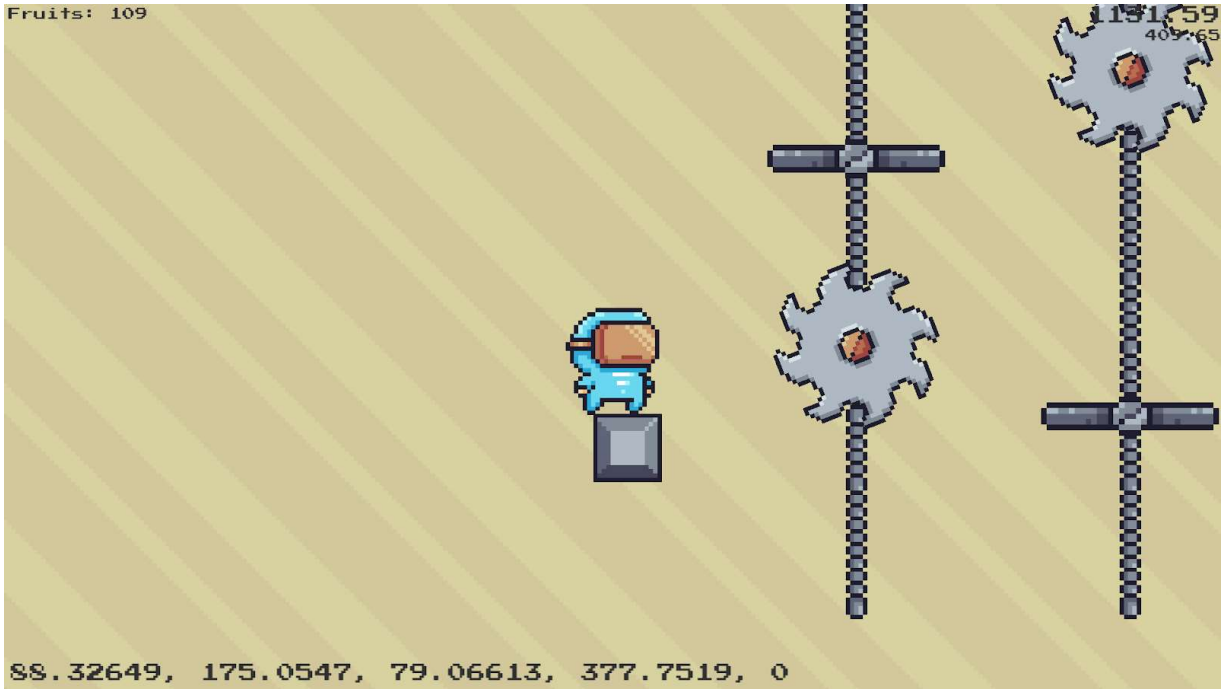

Fruits: 75
686.52
343.82
88.32649, 175.0547, 79.06613, 0, 0

Level 4


Fruits: 75
722.15
1.54
88.32649, 175.0547, 79.06613, 377.7519, 0


Fruits: 80
772.68
51.93
88.32649, 175.0547, 79.06613, 377.7519, 0

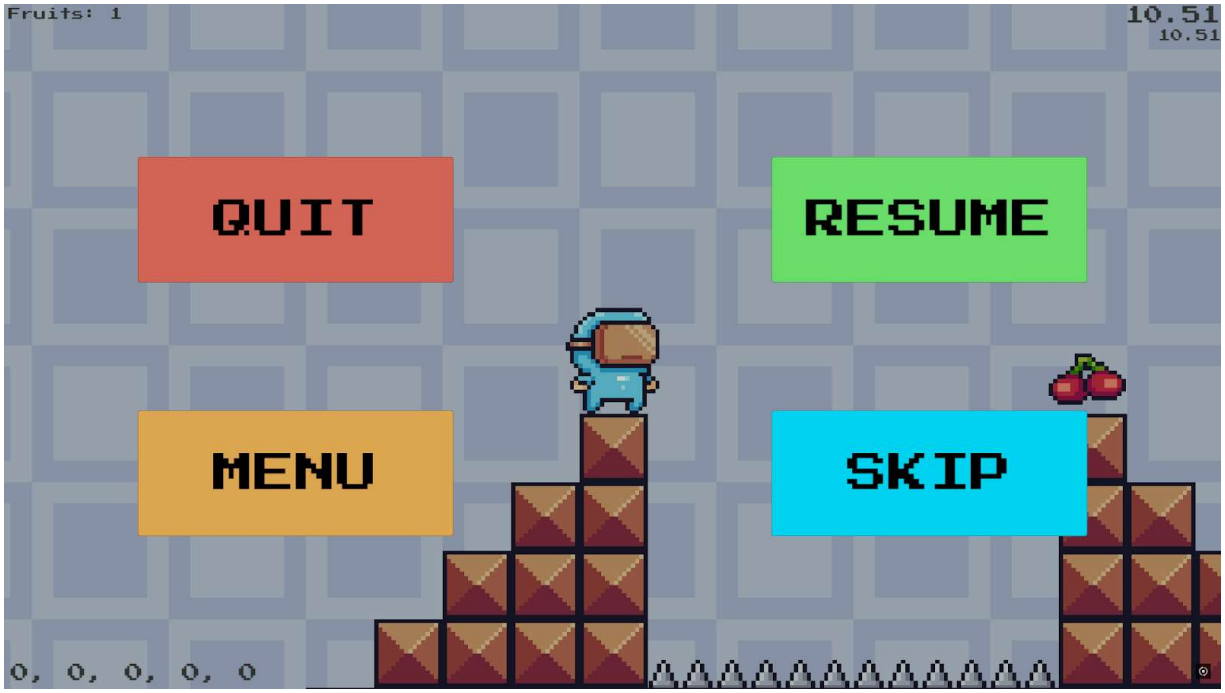

Fruits: 89
885.38
164.31
88.32649, 175.0547, 79.06613, 377.7519, 0



Level 5

End screen with API quote (online)

End screen with a random default quote (offline)



Pause menu

```
**************************************************************************************************
17/01/2025
Level 1:    88.32649
Level 2:    175.0547
Level 3:    79.06613
Level 4:    377.7519
Level 5:    429.7385
Total time:    1151.76
Times for levels 1-5 respectively:    88.32649, 175.0547, 79.06613, 377.7519, 429.7385
Times sorted by shortest to longest:    79.06613, 88.32649, 175.0547, 377.7519, 429.7385
**************************************************************************************************
```

Text file output

Testing

Build 1 Test 28/09/2024

This is a complete test of the start screen, levels 1-4, and the end screen. Completed before level 5 development.

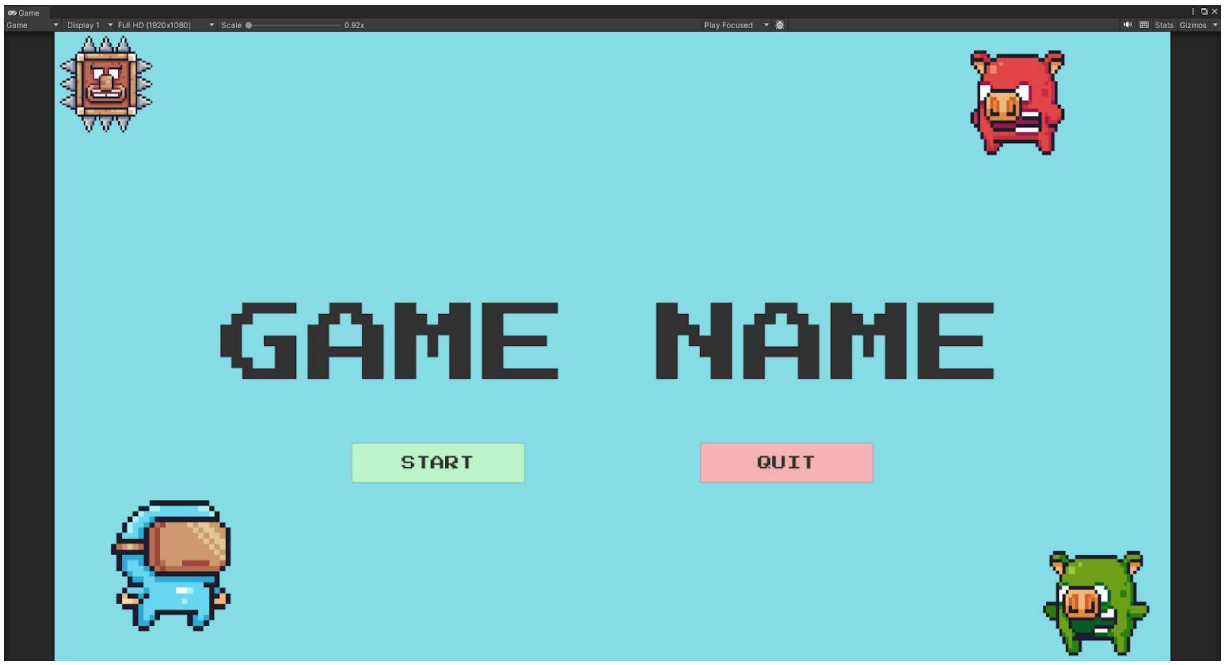
Part of game	Expected result	Actual result	Solution
Start screen	• Start button works • Quit button works • UI formatted correctly	• Start button works • Quit button works • UI does not scale to fullscreen, only same size as when played in editor	• Scale UI from reference points (corners of screen etc) • Force 1920x1080 resolution, not just aspect ratio
Main levels	• Textures complete and set to correct layer e.g ground, background • Player movement and animation works • Camera follows player • Traps can kill player and level restarts	• Textures complete and set to correct layer e.g ground, background • Player movement and animation works • Camera follows player • Traps can kill player and level restarts	• Increase font size (roughly double) • Make disappearing platforms return a short time after they are destroyed • Add fruits/collectibles to level 4

	• Levels can be completed • UI scales and functions correctly • Audio plays at correct times	• Levels can be completed • UI in corners is too small when viewed in fullscreen, but position is correct • Audio plays at correct times • Levels cannot be completed if you fall from a disappearing platform, as they do not come back • Level 4 is missing fruits/collectibles	
End screen	• Restart button works • Quit button works • UI formatted correctly	• Restart button works • Quit button works • UI does not scale to fullscreen, only same size as when played in editor	• Scale UI from reference points (corners of screen etc) • Force 1920x1080 resolution, not just aspect ratio

Test screenshots

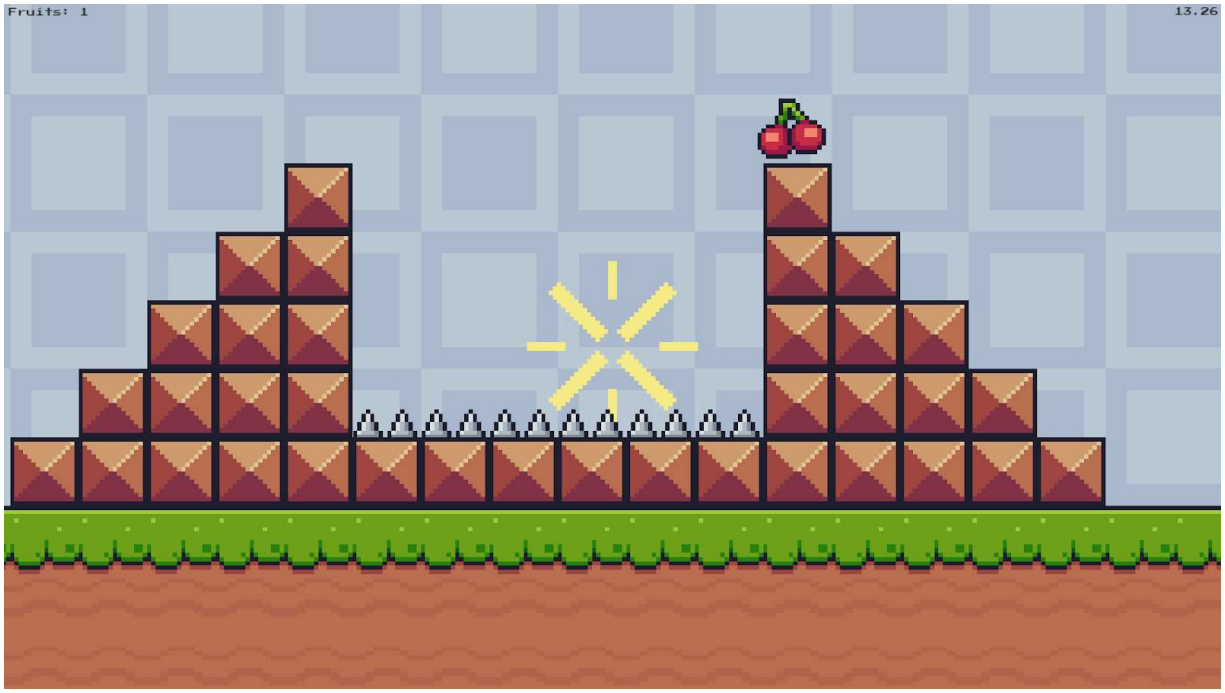
Start screen

Before - Not scaled to fullscreen

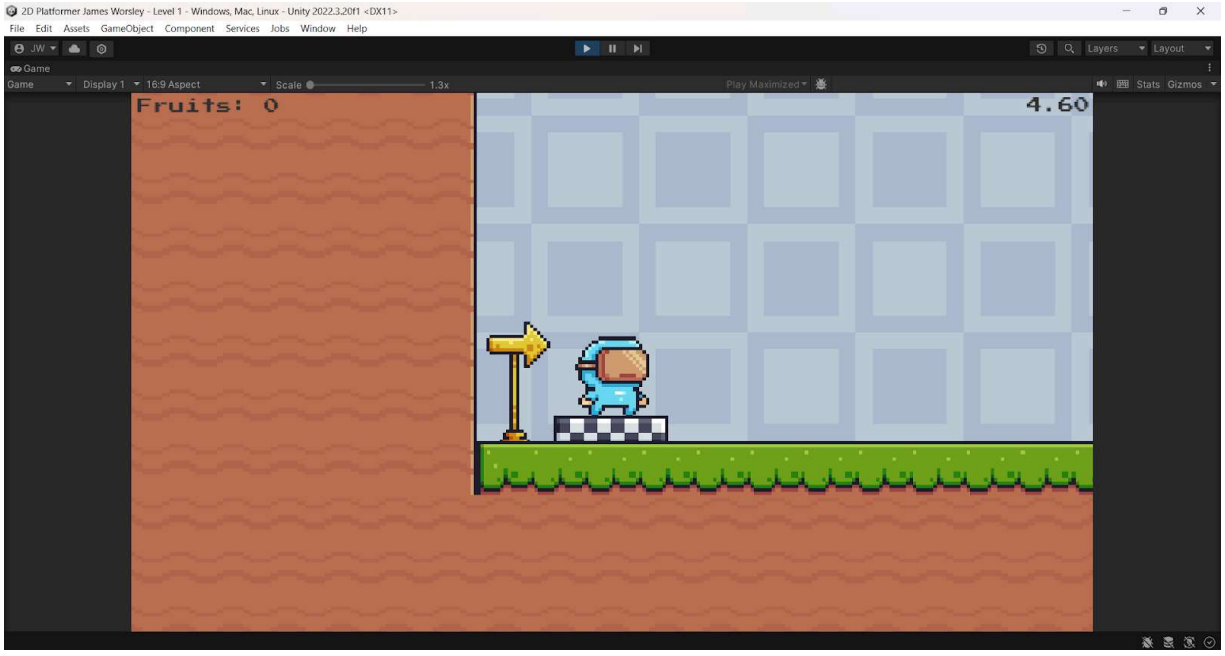


After - Scaled larger

Main levels



Before - UI is too small when viewed in fullscreen (fruit counter and timer in corners)



After - UI text size increased so it is easier to read in fullscreen

End screen

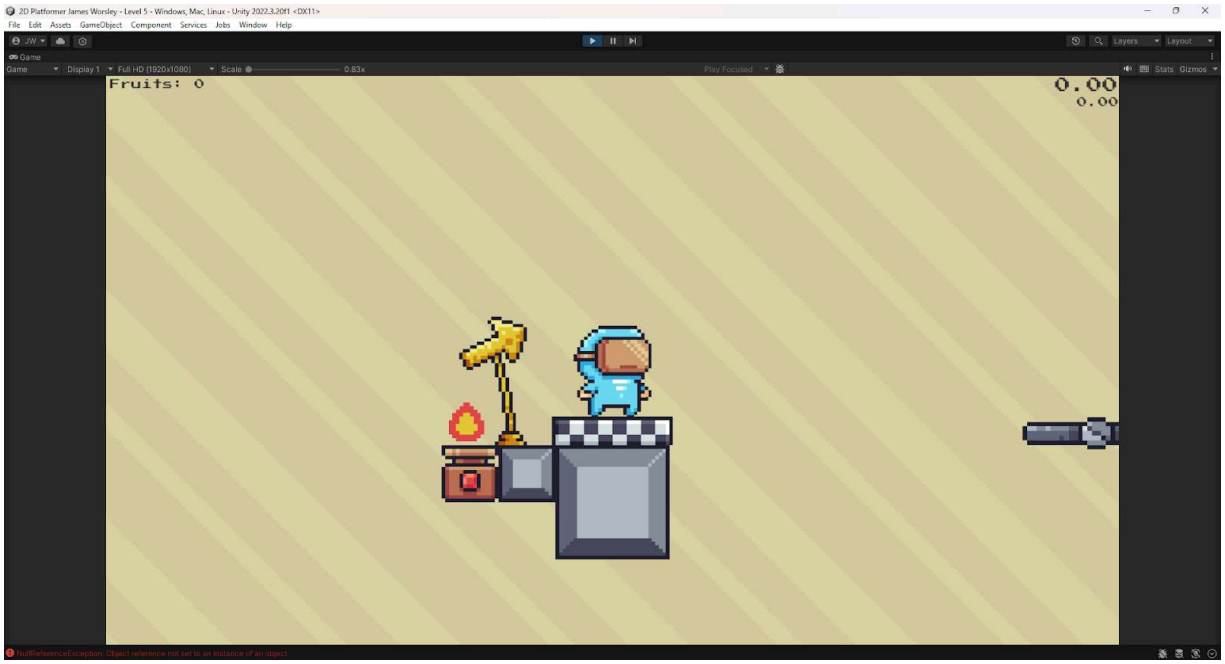
Before - UI not scaled to fullscreen

After - Scaled larger

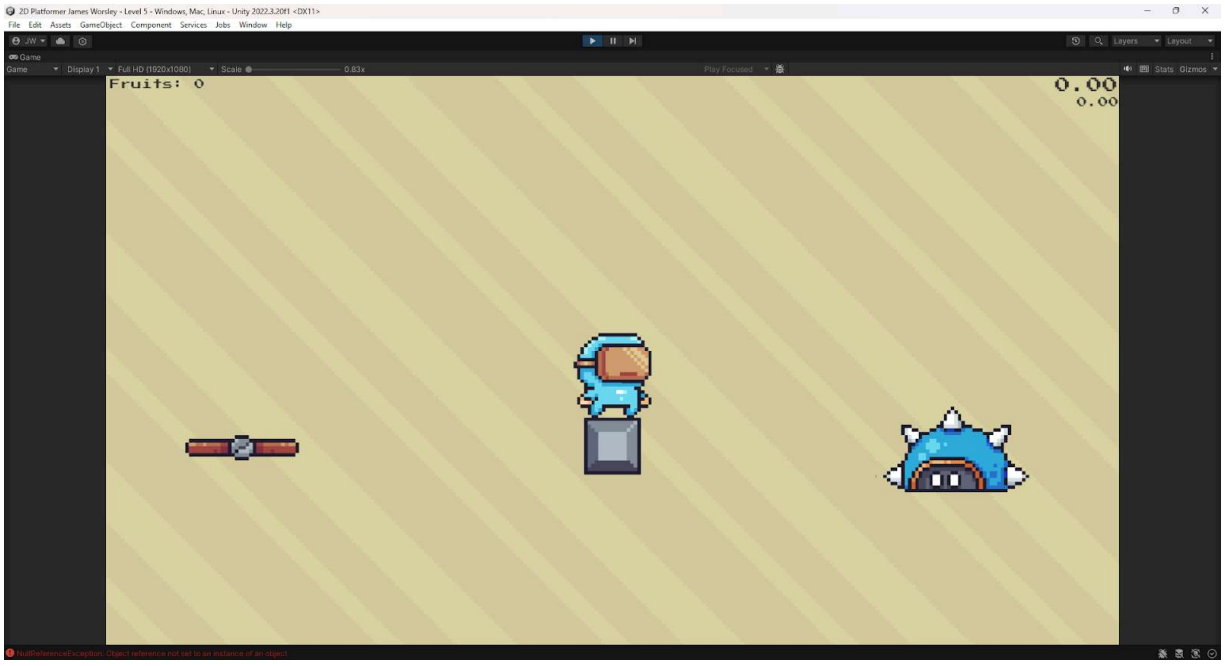
Level 5 Test

Since level 5 has now been added since the last full game test, I will test it on its own so it will be ready for the next full game test.

Part of Level	Expected result	Actual result	Solution
Ease of completion	Level is difficult, but can be completed easily enough that it can be done in one playthrough of the game	• Level completion feels very tense, as intended as the last level • Each platforming section is possible to be completed in isolation • Was not able to beat the entire level in one go, will be worse after also playing through levels 1-4 first	• Reduce difficulty of the more difficult section (see areas bellow) e.g. spike turtle jumping section • Add a skip button (potentially in a pause menu) making sure that this cannot be abused to reach the top of a leaderboard, should I implement a leaderboard • Implement a rewind feature
UI elements	Fruit counter and timers work	• Neither work while playing just level 5 in the Unity editor because they are carried over all the way from level 1, so will only work if the game is started from level 1	Check during a full game test whether or not UI elements carry over and function correctly to level 5



Start of level 5



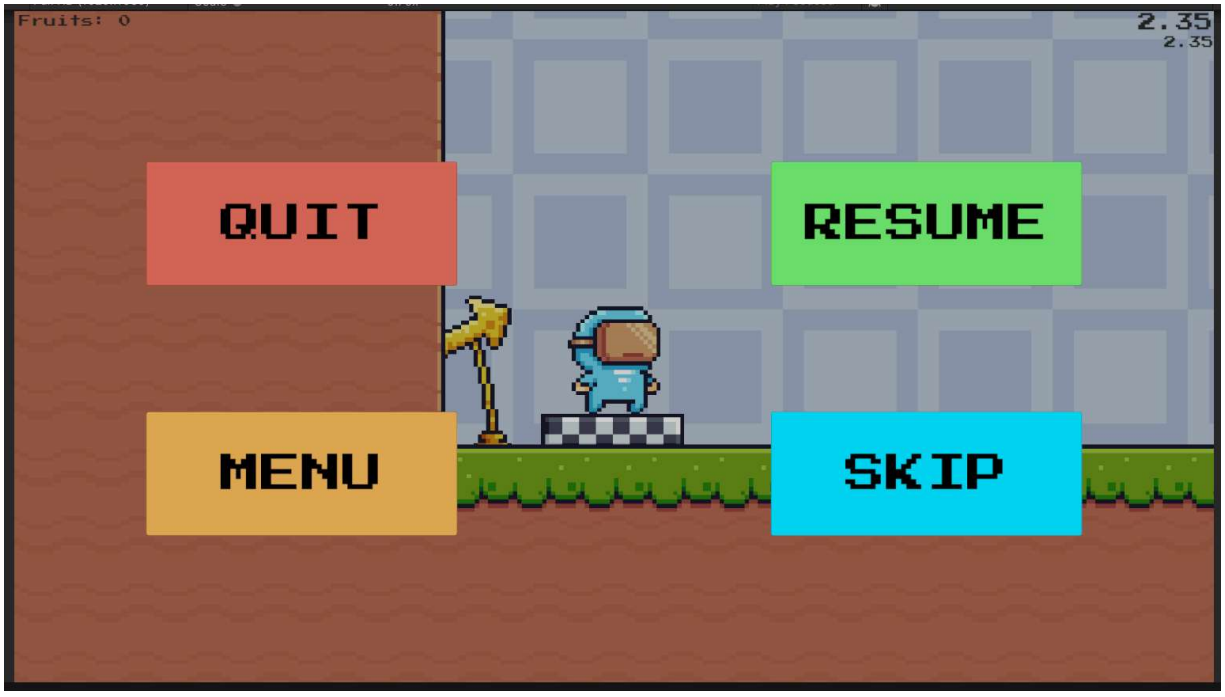
Spike turtle section (right)

Taken 1 spike turtle off the end and converted the new space to be a part of the falling spike head section after. Having removed this section, I was able to complete the level from start to finish. I may still add more assists to make level completion easier for less experienced players, possibly a checkpoint system of some kind utilising stacks.

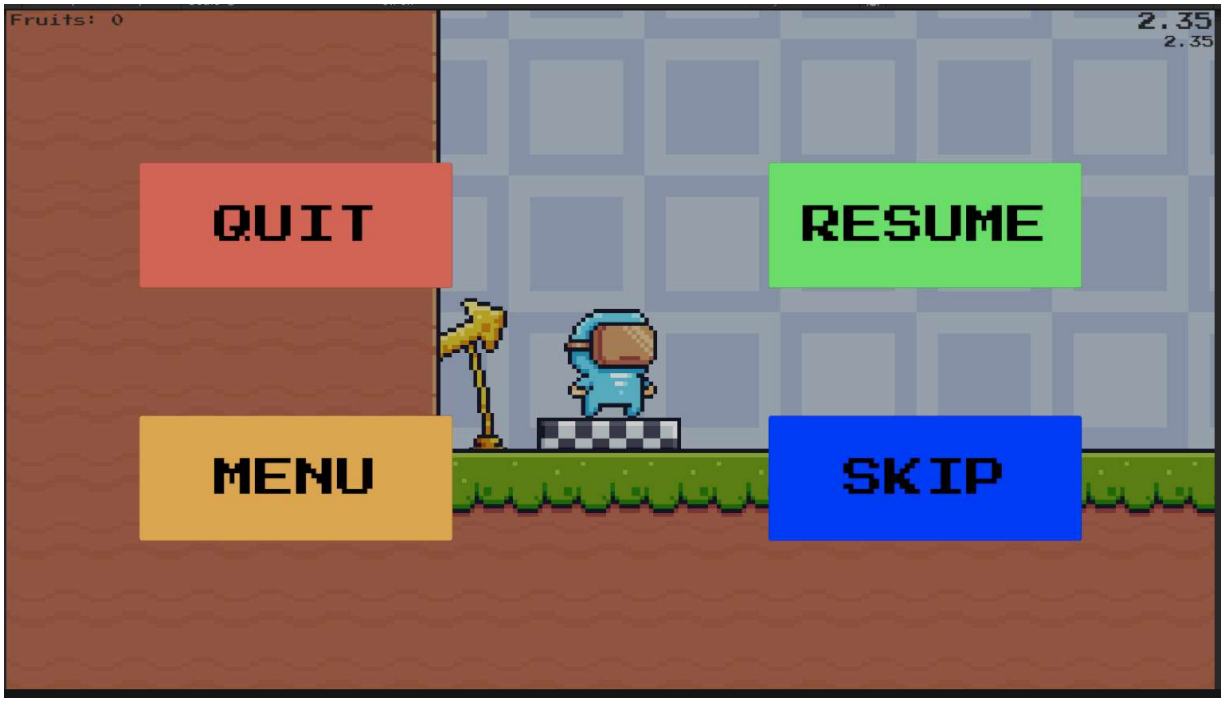
Pause Menu Test

Part of Menu	Expected result	Actual result	Solution
Buttons	• Buttons perform their corresponding function (from PauseMenu script) when pressed • Colour of button changes when hovered over	• Buttons perform their corresponding function (from PauseMenu script) when pressed • Colour of button changes when hovered over	None
Changing between levels	• Pause menu works on any level	• Pause manu works on level 1 (where I	• Do not destroy pause menu on level

	• When level is skipped, game resumes from the start of the next level	designed it) • Pause menu does not work on levels 2-5 • When a level is skipped, the game is still paused	completion, make it carry over with the other UI • Call Resume() function from within the SkipLevel() function level has been incremented
Graphic design	• Each button has a unique, distinct colour • Pause menu is dimmed compared to normal gameplay	• Each button has a unique, distinct colour • Pause menu is dim compared to normal gameplay, but the change is not faded	• Animate the colour of the PauseMenu panel to fade/dim over a short period of time



Pause menu post solution



Colour of button changes when it is hovered over

Build 2 Test 12/11/2024

This is a complete test of the start screen, levels 1-5, the end screen and pause menu. Completed after level 5 and pause menu development.

Part of game	Problems	Solutions
Start screen	None	None
All main levels	• On death, a second timer appears and the two overlap • All levels have the same music	• Currently when the player dies, the persistent UI script creates another timer etc and after 2 deaths it breaks the pause menu. To fix I have put the pause menu on a separate canvas and taken account of the increased size of the array in the persistent UI script. • Change music on

		levels 2-5
Level 1 specific	None	None
Level 2 specific	• Fruit counter doesn't work • Fruit collection has no sound effect	• Create a new simpleton script which counts the number of fruits collected and saves them at the end of each level, fruits is changed back to the saved fruits at the start of each level • Add fruit audio source to ItemCollector script in Unity editor
Level 3 specific	• Fruit counter doesn't work • Fruit collection has no sound effect	Same as above
Level 4 specific	• Fruit counter doesn't work • Fruit collection has no sound effect	Same as above
Level 5 specific	• Fruit counter doesn't work • Fruit collection has no sound effect	Same as above
End screen	None	None
Pause menu	• Does not work past level 1 • Timer does not reset to 0 after level is skipped	• Fixed by solution to timer issue in main level test • Get the Main Menu script to tell the Timer script that the level has been skipped, using a public bool, similar to how the finish script does. Could put the local timer on a separate canvas that is not kept throughout all levels.

Recall Test

Part of feature	Problems	Solutions
Set recall point	None	None
Trigger recall	• Error occurs when recall button is pressed but stack of X and Y coordinates is empty	• Only allow recall attempt when the count of the stacks doesn't equal 0
General use	• No indication that recall point had been set or recall has been triggered beyond log debug	• Play sounds for setting recall point/triggering recall

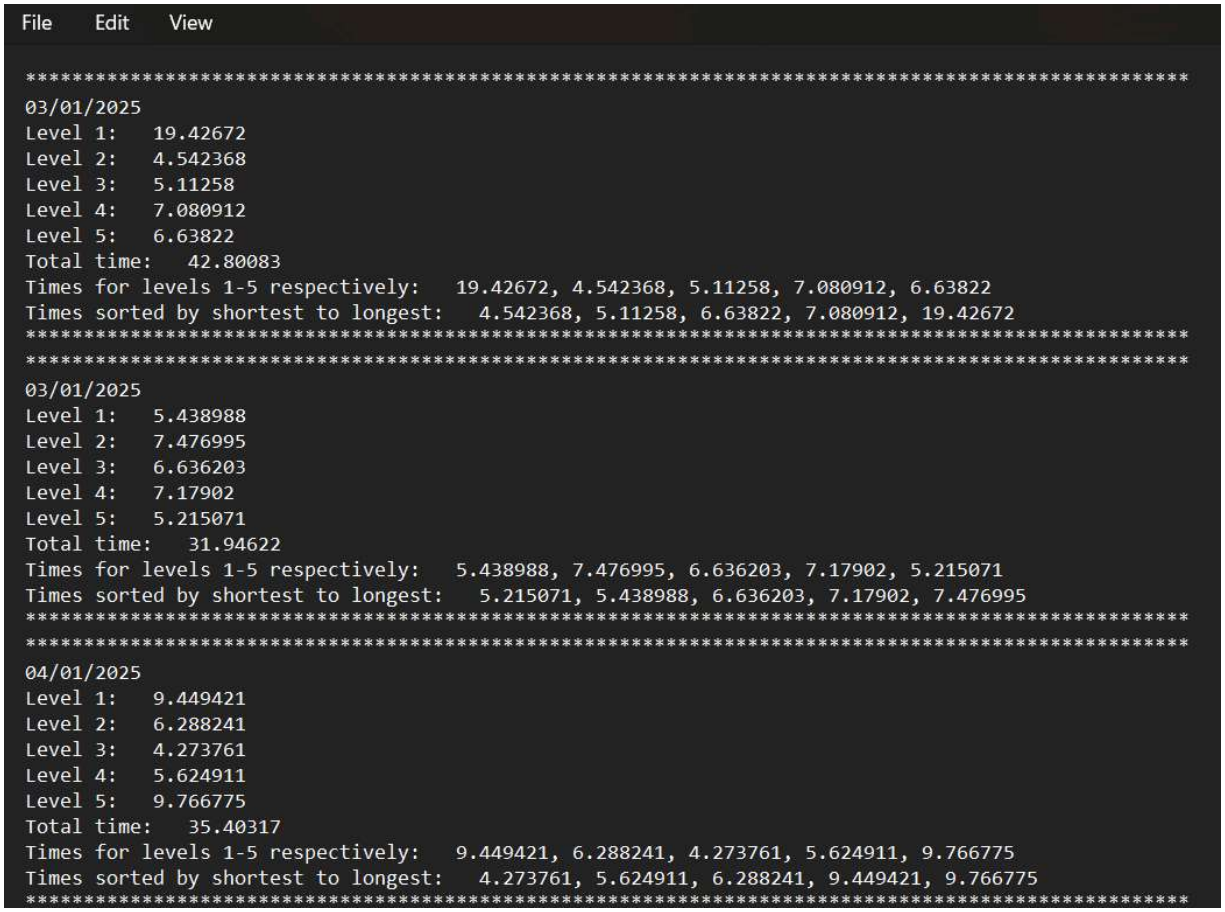
Text File and Times Array Test 1

Issues	Solutions
Text is all on 1 line	Append Environment.NewLine as well as the strings so at the end of each line being written, the next one is started
Text is written repeatedly upon update	Ensure the text is only written once by setting a new boolean to true if the file has been written and only write to the file if this boolean is false.
Times don't show properly on game UI	Don't use ToString() on an array, instead use string.Join() to concatenate the times to a string individually so they can be displayed properly

Text File and Times Array Test 2

Part of feature	Problems	Solutions
Merge sort	None	None

Formatting	Dividing *** lines are not as long as array output	Double length of *** lines



Text file contents post solution.

API Import Test

Part of feature	**Problems**	**Solutions**
Formatting	• Text box is too small to display the entire quote. • Quote and author have no space between.	• Increase the size of the text box. • Add “ - ” in between the quote and the author when writing the API Text.
(Theoretical). The API service could become discontinued.	(Theoretical). API request would return no quote.	(Theoretical). Add a separate default quote if the API request returns nothing.

Before Solution

Text box too small

No space between quote and author.

After Solution

Space added between quote and author and text box made larger.

```C/C++
if (webRequest.result == UnityWebRequest.Result.ConnectionError ||
webRequest.result == UnityWebRequest.Result.ProtocolError)
{
    quoteText.text = defaultQuotes[Random.Range(0, defaultQuotes.Length)];
}
```

This code should ensure that if the API import fails, a quote from the predefined array:

```C/C++
private string[] defaultQuotes = {"Have a great day!", "Well played!", "Play
again soon!"};
```

An element of this list will be displayed on the end screen instead.

Build 3 Test 04/01/2025

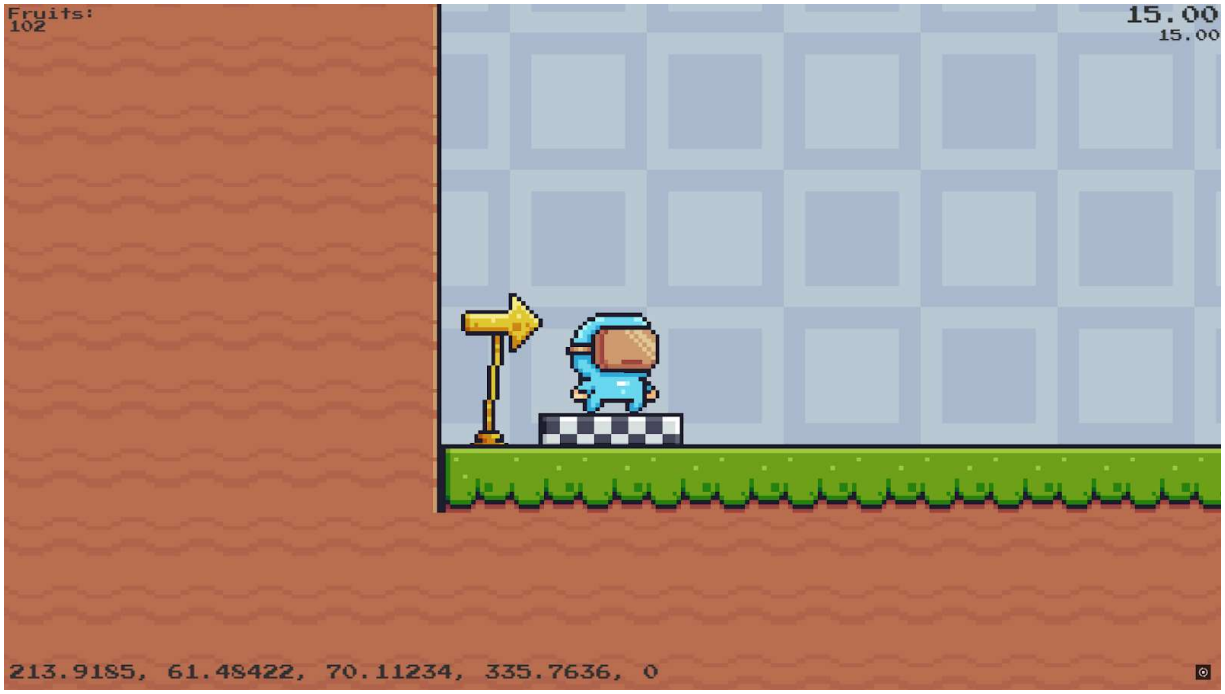
This build was created after the addition of recall, times array, file saving, merge sort and API import. The game is complete in terms of features, but needs some problems to be fixed.

Part of game	Problems	Solutions
Start screen	None	None
Inputs	• Can't pause the game from controller • Can't move up with "W" key in flying item because it is assigned to jump	• Modify pause menu script to allow player to pause the game using the option button on the controller • Change "W" key to move up instead of jump
General gameplay	• Spawn set and recall sounds play on player ever level start • Jump sound also play on some levels starting	• Uncheck "Play On Awake" on all audiosources on player that are not background music • Uncheck "Play On Awake" on all audiosources on trap prefabs that are not background music
Level design	• Background on level 5 is too small so it runs out before the invisible death plane • Death plane on level 5 is too thin vertically so can be passed through if player teachers a great enough velocity	• Increase the size of the level 5 background • Increase the vertical width of death plane to 50 instead of 1
UI	• Fruit counter text box is too small • Times array and fruit counter do not reset upon game restart • Time counted as 0 for skipped level so total time is faster than	• Increase size of fruit counter text box • Reset times array to 0,0,0,0,0 upon restart or upon reaching the start screen • Could add a time penalty for skipping

	playthroughs when levels are not skipped	levels
Pause menu	None	None
End screen	Colour on non-highlighted buttons are slightly too dim	Make the non-highlighted colour of the buttons more bold

```
**********************************************************************************************************
04/01/2025
Level 1:    213.9185
Level 2:    61.48422
Level 3:    70.11234
Level 4:    335.7636
Level 5:    0
Total time:    965.9287
Times for levels 1-5 respectively:    213.9185, 61.48422, 70.11234, 335.7636, 0
Times sorted by shortest to longest:    0, 61.48422, 70.11234, 213.9185, 335.7636
**********************************************************************************************************
```

Text file output



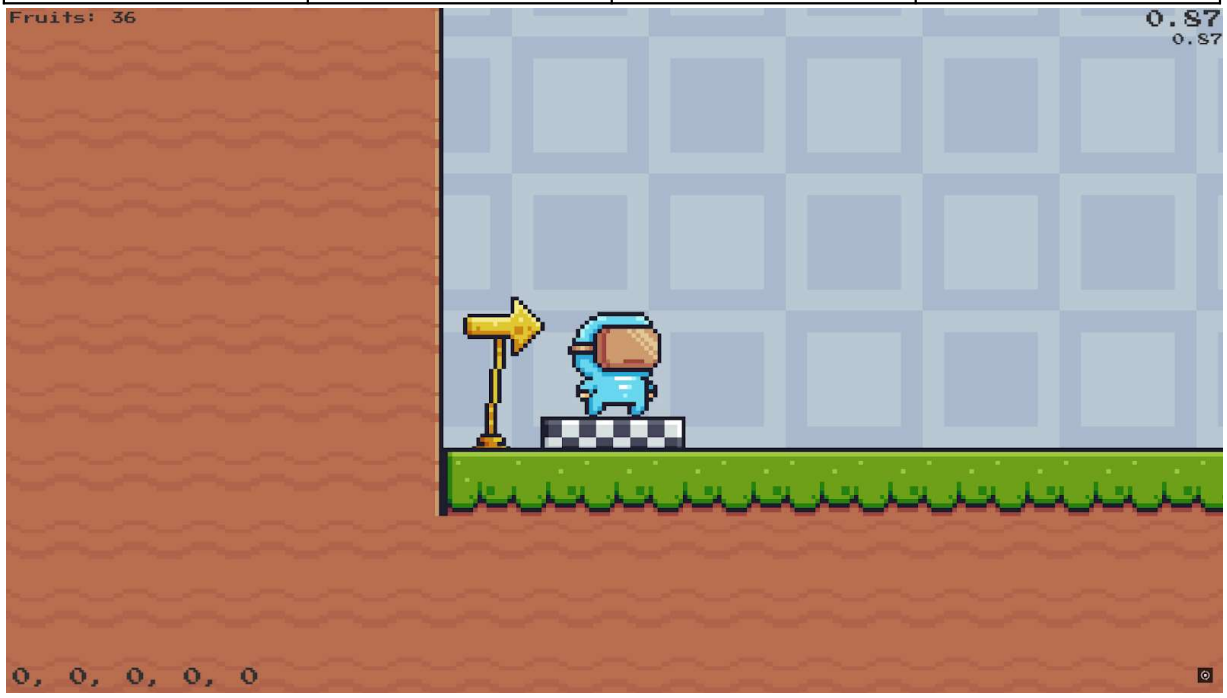
Fruit counter and times array not reset after game restart, fruit counter text box is too small

Build 4 Test 10/01/2025

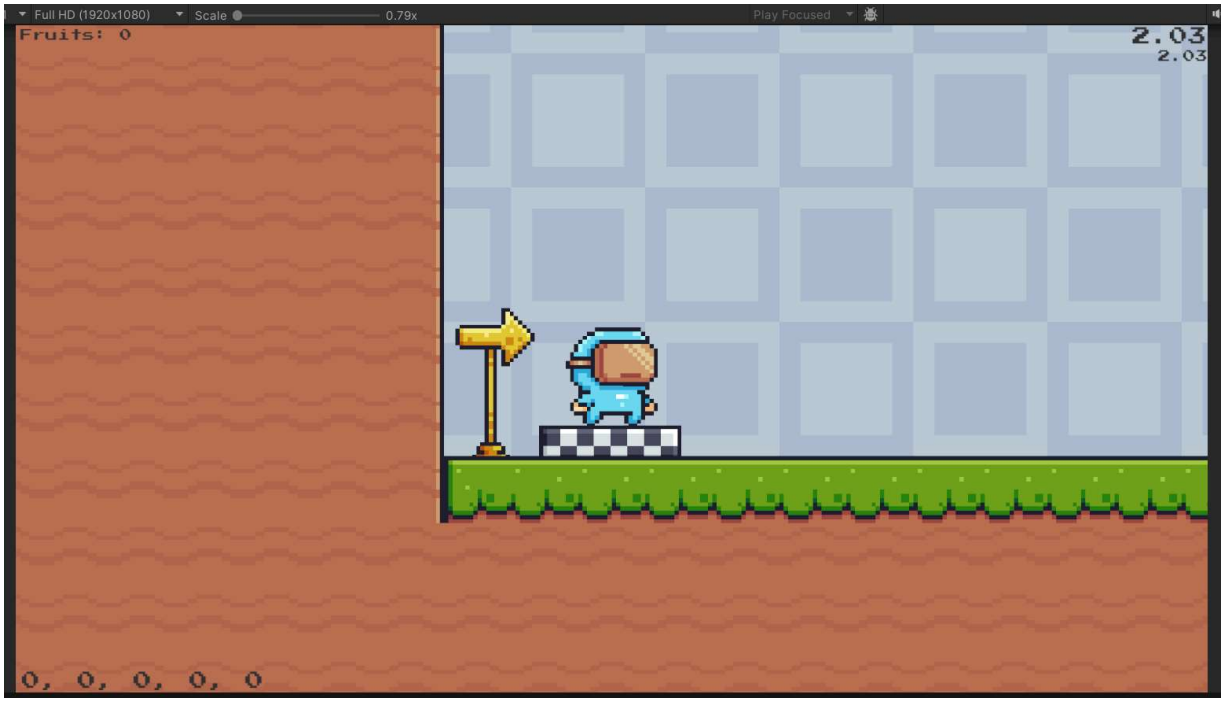
Part of game	Expected result	Actual result	Solution
Start screen	• Quit button works • Start button loads level 1	• Quit button works • Start button loads level 1	None
Inputs	• Keyboard input functions correctly • Controller input functions correctly	• Keyboard input functions correctly • Controller input functions correctly	None
Level design	• No accidentally placed terrain • Levels are completable and follow general difficulty gradient • Background and over-background placed correctly	• No accidentally placed terrain • Level 5 has a very tight section with the falling spike heads • Levels follow general difficulty gradient • Background and over-background placed correctly	• Move falling spike heads up a fraction
UI	• Timers work properly • Times saved and displayed to times array at end of each level • Fruit counter tracks fruits collected accurately	• Timers work properly mostly • Times do not reset on level 1 if the game has been restarted through the pause menu • Times saved and displayed	• Always set local and global times to 0f at the start of level 1 • Always set fruit counter to 0 at the start of level 1

		to times array at end of each level • Fruit counter does not reset upon restart	
Audio	• Audio plays at the correct time • Correct audio plays on each level • Only background music plays upon start	• Audio plays at the correct time • Correct audio plays on each level • Only background music plays upon start	None
Pause Menu	• All buttons work • Time stops while paused • Game screen dims • No problems when changing scene	• All buttons work • Time stops while paused • Game screen dims • Game is still paused when the game is started having returned to menu using the pause button	• Call the Resume() subroutine in the LoadMenu() subroutine
Recall system	• Audio plays to show spawn is set or recall is taking place • Spawn point can only be set on ground • Player's position is correctly set upon recall • Stack can only be popped from if it is not empty	• Audio plays to show spawn is set or recall is taking place • Spawn point can only be set on ground • Player's position is correctly set upon recall • Stack can only be popped from if it is not empty	None
File saving	• Times save to	• Times save to	None

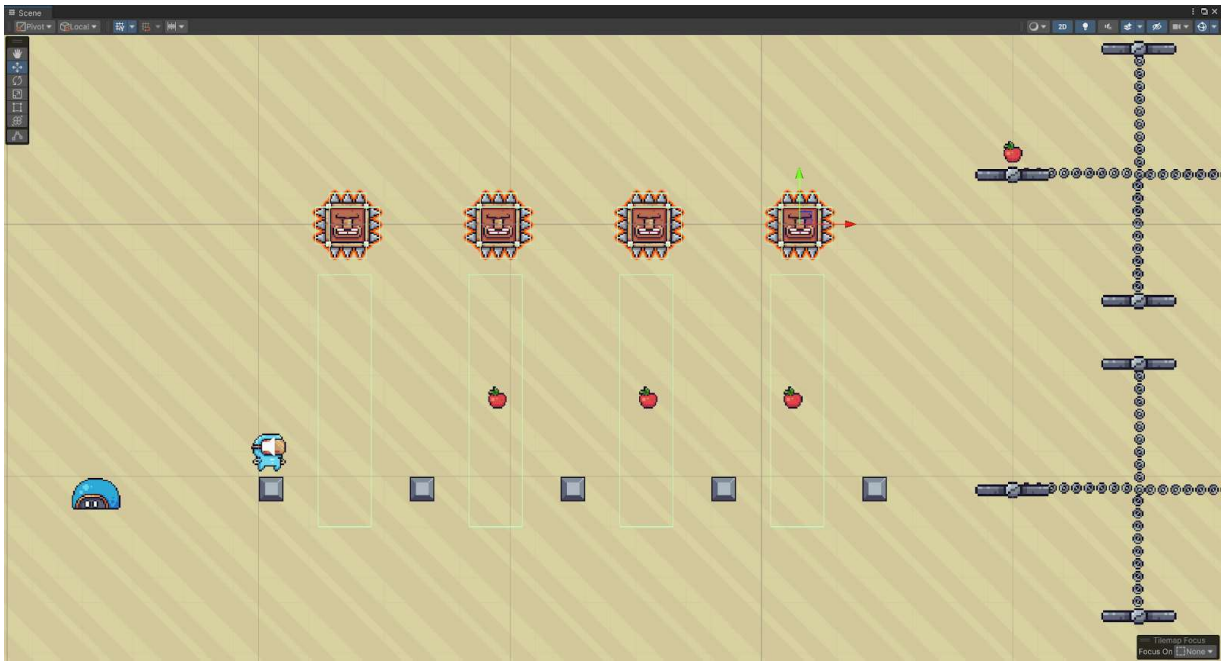
	text file • Fruits save to text file • Merge sort gives correct output • Formatting the same for each game completion	text file • Fruits save to text file • Merge sort gives correct output • Formatting the same for each game completion	
End screen	• Quit button works • Times array, global and local time and fruit count all return to 0 upon restart	• Quit button works • Times array, global and local time return to 0 upon restart • Fruit counter does not return to 0 upon restart	• Always set fruit counter to 0 at the start of level 1



Fruits counter still at 36 from the previous playthrough, despite being at the start of level 1. Timers (top right) and times array (bottom left) have been reset correctly.

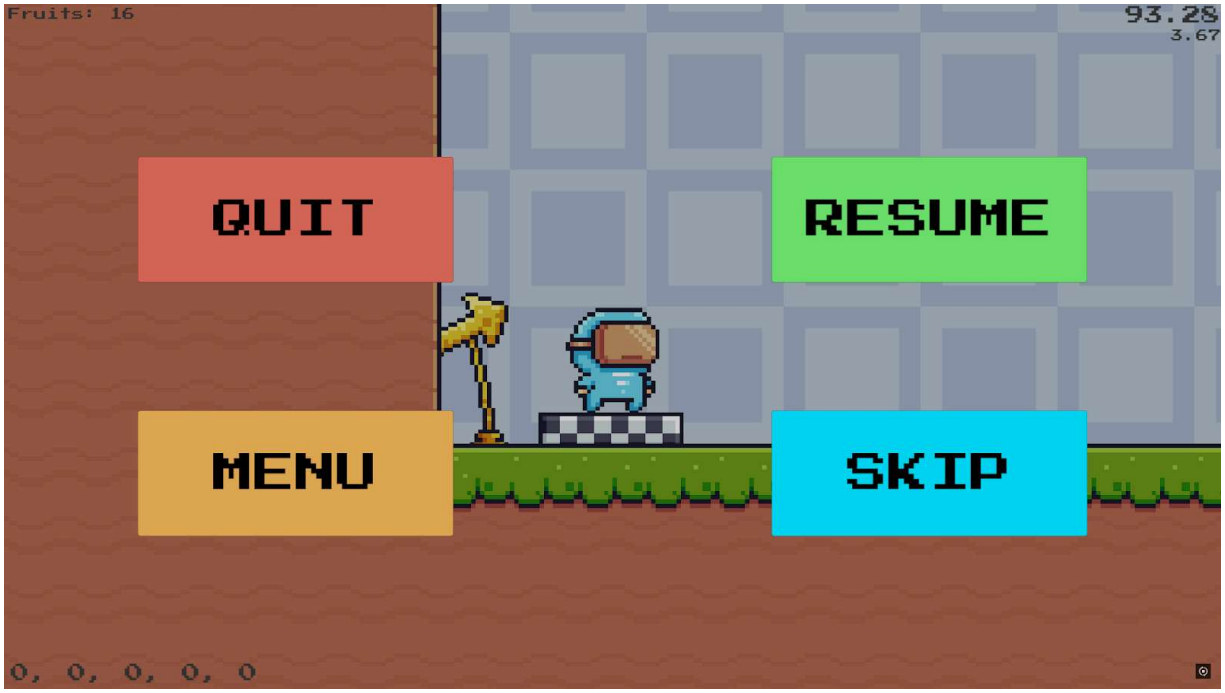


Fruit counter always set to 0 at the start of level 1



Falling spike heads too low

Spike heads moved up slightly



Game still paused after starting the game having returned to the main menu in the pause menu. Fruit counter and timers also not reset to 0.

Offline API Import Test

This is a test of the API import on the end screen whilst my laptop is not connected to the internet to simulate what would happen if the API could not be fetched. This will test the random selection of a quote from the defaultQuotes array. This exception handling is needed in case the API gets shut down in the future.

Part of feature	Expected outcome	Actual outcome	Solution
Use of default quotes array	Random quote from defaultQuotes array is used instead of quote from API	Random quote from defaultQuotes array is used instead of quote from API	None
Formatting	Quote is displayed entirely at a logical font size in Press Start font	Quote is displayed entirely at a logical font size in Press Start font	None

Quote at the bottom is from the defaultQuotes array as intended

Another quote from the defaultQuotes array

Final Build Test (Build 5) Test 17/01/2025

This is the test of the final game in its entirety[22]. All features are now present and working including file saving, merge sort, recall and API import. The game has a start screen, followed by levels 1-5 and closing with the end screen. The game can be paused at any time during levels 1-5, activating the pause menu which has options to resume, quit the game, return to the start screen/menu and skip the current level.

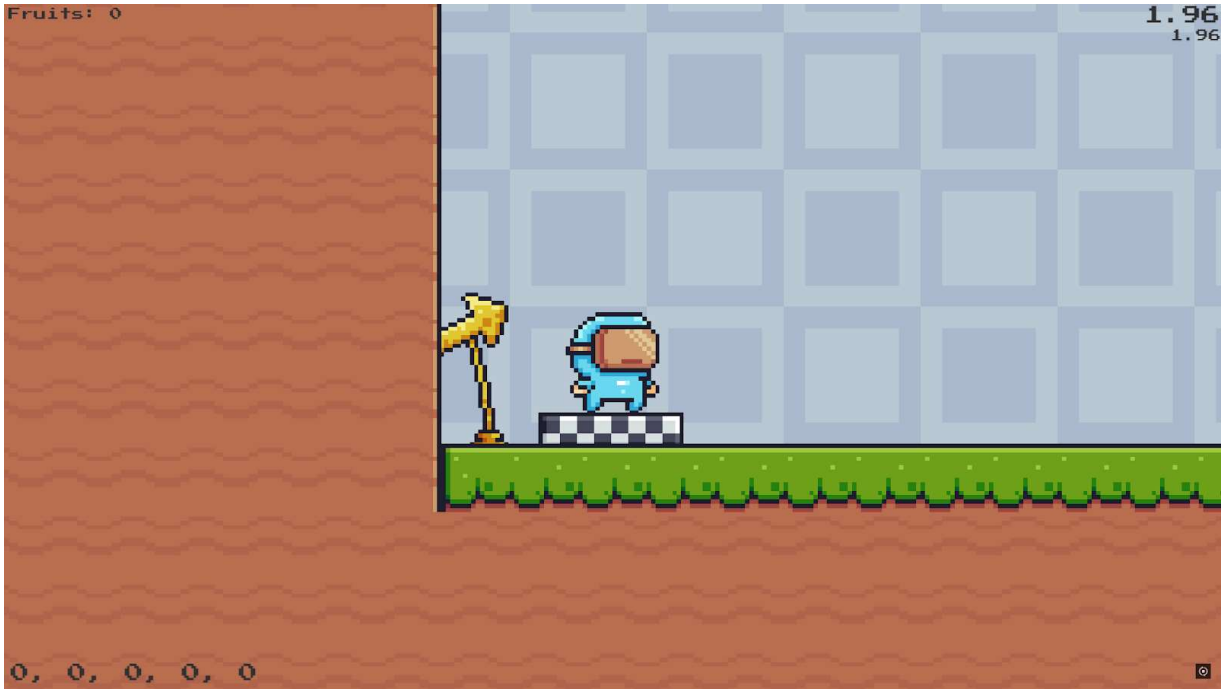
Part of game	Expected result	Actual result	Solution
Start screen	• Quit button works • Start button loads level 1	• Quit button works • Start button loads level 1	None
Inputs	• Keyboard input functions correctly • Controller input functions correctly	• Keyboard input functions correctly • Controller input functions correctly	None

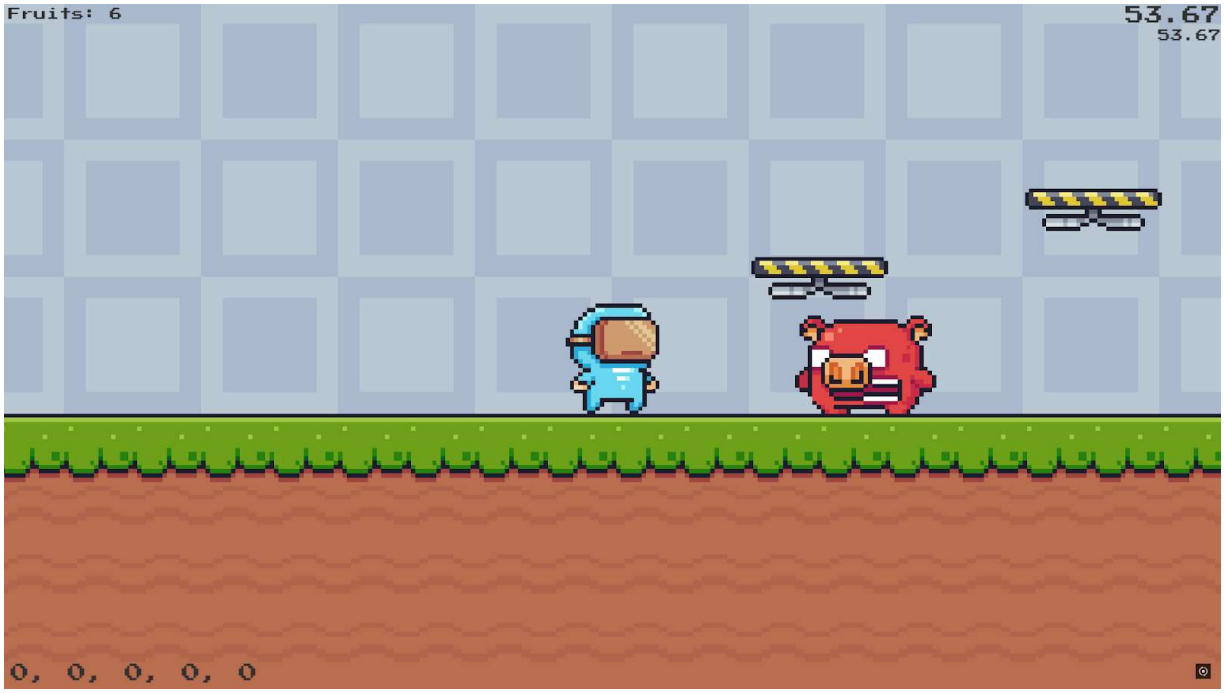
[22] Video of test https://youtu.be/el9u33uxcCE Uploaded 28/02/2025

Level design	• No accidentally placed terrain • Levels are completable and follow general difficulty gradient • Background and over-backgrou nd placed correctly	• No accidentally placed terrain • Levels are completable and follow general difficulty gradient • Background and over-backgrou nd placed correctly	None
UI	• Timers work properly • Times saved and displayed to times array at end of each level • Fruit counter tracks fruits collected accurately	• Timers work properly • Times saved and displayed to times array at end of each level • Fruit counter tracks fruits collected accurately	None
Audio	• Audio plays at the correct time • Correct audio plays on each level • Only background music plays upon start	• Audio plays at the correct time • Correct audio plays on each level • Only background music plays upon start	None
Pause Menu	• All buttons work • Time stops while paused • Game screen dims • No problems when changing scene	• All buttons work • Time stops while paused • Game screen dims • No problems when changing scene	None
Recall system	• Audio plays to	• Audio plays to	None

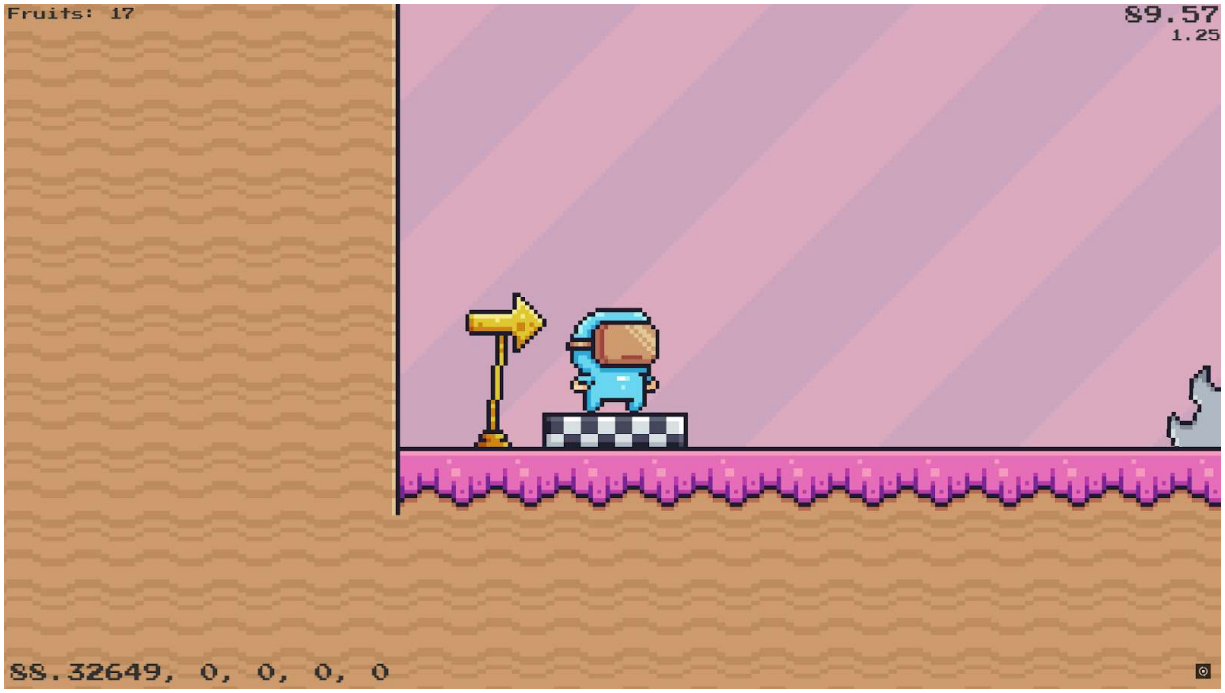
	show spawn is set or recall is taking place • Spawn point can only be set on ground • Player's position is correctly set upon recall • Stack can only be popped from if it is not empty	show spawn is set or recall is taking place • Spawn point can only be set on ground • Player's position is correctly set upon recall • Stack can only be popped from if it is not empty	
File saving	• Times save to text file • Fruits save to text file • Merge sort gives correct output • Formatting the same for each game completion	• Times save to text file • Fruits save to text file • Merge sort gives correct output • Formatting the same for each game completion	None
End screen	• Quit button works • Restart button works • Times array, global and local time and fruit count all return to 0 upon restart • API import formats correctly • Default quote randomly displayed if API import fails (test with laptop not connected to the internet)	• Quit button works • Restart button works • Times array, global and local time and fruit count all return to 0 upon restart • Default quote randomly displayed if API import fails	None

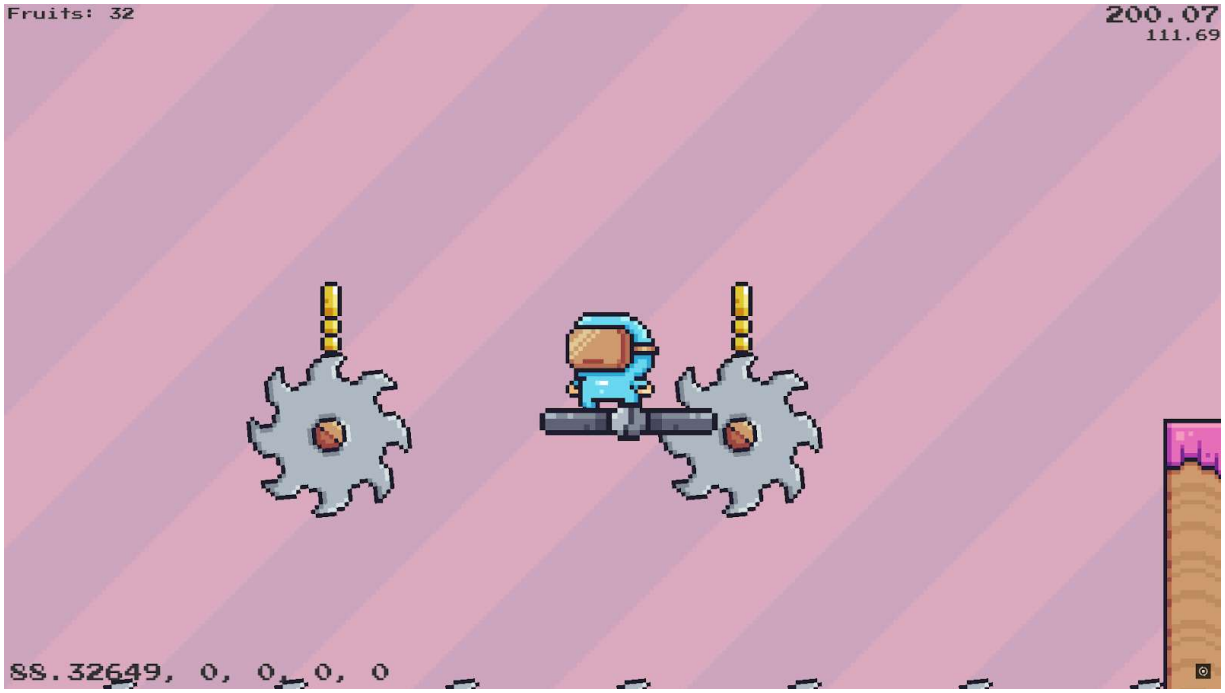
Start screen





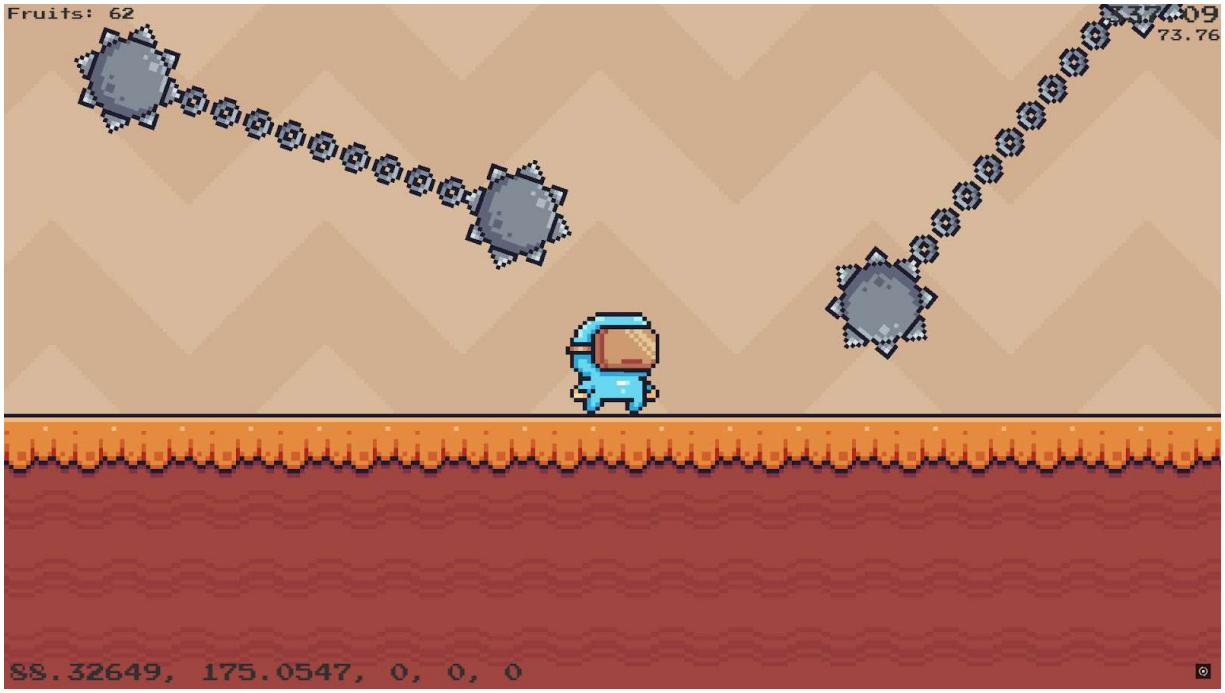
Level 1




Fruits: 32
200.07
111.69
88.32649, 0, 0, 0, 0

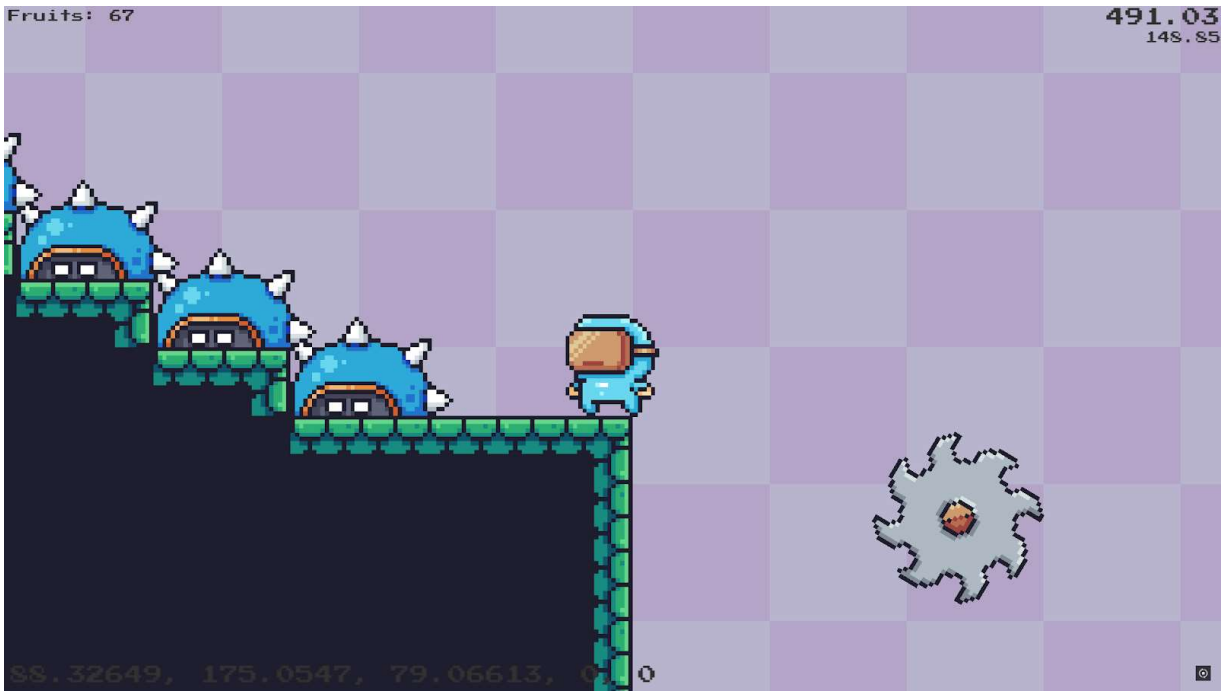
Level 2


Fruits: 43
264.60
1.16
88.32649, 175.0547, 0, 0, 0


Fruits: 62
88.32649, 175.0547, 0, 0, 0

Level 3


348.09
88.32649, 175.0547, 79.06613, 0, 0


Fruits: 67
491.03
148.85
88.32649, 175.0547, 79.06613, 0, 0


Fruits: 72
584.25
241.96
88.32649, 175.0547, 79.06613, 0, 0

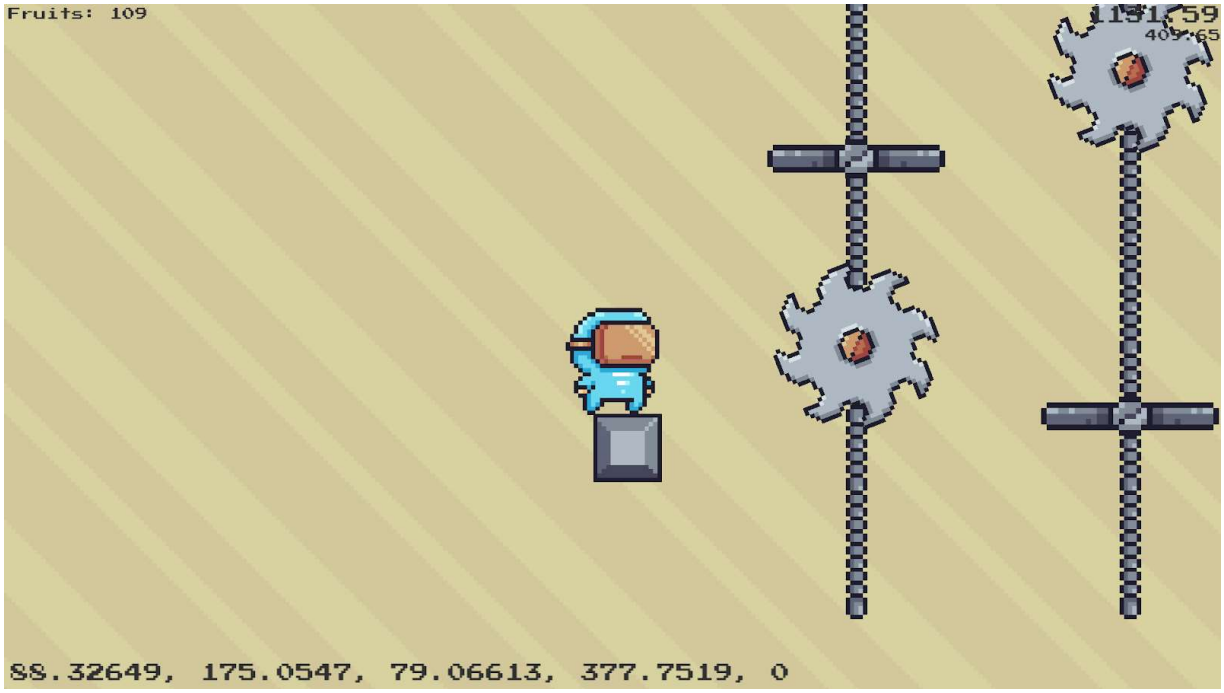

Fruits: 75
686.52
343.82
88.32649, 175.0547, 79.06613, 0, 0

Level 4


Fruits: 75
722.15
1.54
88.32649, 175.0547, 79.06613, 377.7519, 0


Fruits: 80
772.68
51.93
88.32649, 175.0547, 79.06613, 377.7519, 0

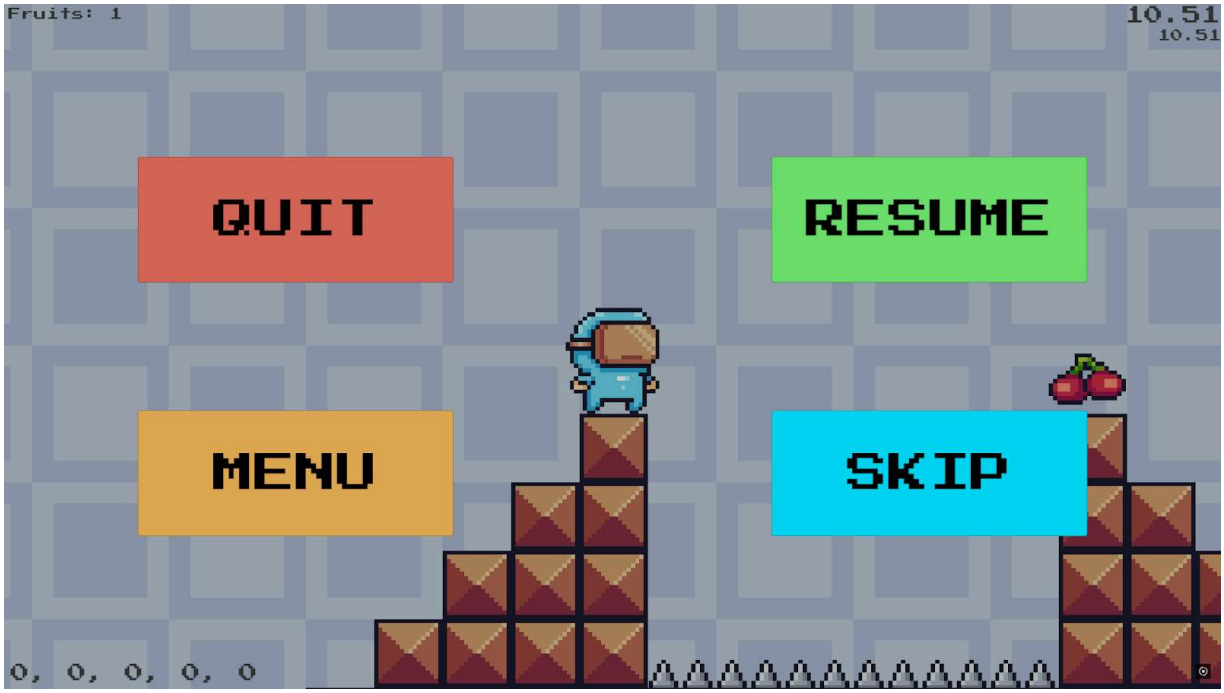

Fruits: 89
885.38
164.31
88.32649, 175.0547, 79.06613, 377.7519, 0



Level 5

End screen with API quote (online)

End screen with a random default quote (offline)



Pause menu

```
****************************************************************************************************
17/01/2025
Level 1:    88.32649
Level 2:    175.0547
Level 3:    79.06613
Level 4:    377.7519
Level 5:    429.7385
Total time:    1151.76
Times for levels 1-5 respectively:    88.32649, 175.0547, 79.06613, 377.7519, 429.7385
Times sorted by shortest to longest:    79.06613, 88.32649, 175.0547, 377.7519, 429.7385
****************************************************************************************************
```

Text file output

Client Test 02/02/2025

This is a test completed by both me and the client (my dad) against the requests made by the client during the questionnaire in the analysis stage. Same build as the final build level 5.

Client Request	Completed?	Possible improvements
Similar to Super Mario World and other older 2D platformers	Yes • The game is focused on tight platforming sequences of increasing difficulty • Tight controls • Both have a 16-bit art style	Add boss fights (e.g fighting Bowser in Super Mario World).
Continuous side scrolling	Yes • Each level scrolls from left to right • Camera follows player's movement • Some levels also have a large vertical element • Camera is never locked, it always scrolls in line with the player	Force the camera to always move independently of the player's movement to increase time pressure.
Moving platforms	Yes • Moving platforms move between waypoints in a list • After completing the list of waypoints, the platform moves back to the start and	None

	repeats • Variations such as rotating platform	
Time element	Yes • Global timer to show time to beat all levels • Local timer to show time to beat individual level • Array of times to see time to beat each level individually • Times saved to text file for direct access • Merge sort sorts times from shortest to longest	Have a timer count down from x seconds and if the player runs out of time they have to restart.
Cloud item (from Super Mario World)	Yes • The flying item replicates the behaviour of the clouds in Super Mario World	None
Disappearing/falling platforms	Yes • Recurring platforms disappear after being stepped on for x seconds, and return after y seconds	Could get platforms to fall after a certain time, but the platforms returning is important to make sure levels are always completable.
Swinging platforms/pendulums	Yes • Swings rotate for 180 degrees • Can adjust direction and speed of swings in editor • Can offset swing patterns between platforms	Make swing carry player.
Falling ceilings	No • Made redundant by the falling spike head • Most levels don't feature a roof	Add falling ceilings.
Checkpoints	Yes	None

	• No physical set checkpoints • Player can set their own checkpoints using the recall system	
16-bit art style	Yes • The assets imported from Pixel Adventure 1 and 2 are in a 16-bit art style	None
Few, longer levels	Yes • Most levels are long, but still very in length • Level 3 is fairly short to add variety • Level 5 is very long to be a final challenge	Add more levels.
Non-rhythm platformer	Yes • Each level has normal background music • Platforming is not based on the music timing	None

See Final Build Test (Build 5) for screenshots.

Evaluation

Original Objectives

Technical Objectives

- Use the Unity game engine and C# programming language
- Implement object oriented programming, including classes and inheritance
- Implement stack operations to allow player to set/recall their position
- Implement list operations for:
    - Waypoint following
    - Times to beat levels
    - Quotes used if API import fails
- Implement file saving (text file and/or SQL) for direct access to save:
    - Date of completion
    - Times for each level
    - Total time
    - Times sorted using merge sort
- Utilise a merge sort to sort level times
- Utilise a recursive algorithm in the merge sort
- Import a web service API and use JSON parsing to give the player a message for completing the game
- Use some form of complex maths
- Use vectors for
    - Player movement
    - Platform movement
    - Trap movement
    - Enemy movement
- Animate game objects
    - Player has movement states for:
        - Running
        - Jumping
        - Falling
        - Idle
        - Death
    - Basic animations for:
        - Enemies (spike turtle more advanced)
        - Traps
        - Platforms
        - Power ups

Gameplay Objectives

- Game meets requirements of client
- Camera follows the player's position
- 16 bit art style, similar to that of Super Mario World
- Start screen
- Main levels
    - Multiple levels
    - Varying length
    - Horizontal vertical scrolling
    - Levels in approximate difficulty order
- End screen with random quote from API
- Allows keyboard, mouse and controller inputs
- Recall system
    - Spawn point can be set (pushed onto stacks)
    - Player can teleport to the last spawn point (pop from stack)
    - Audio plays on spawn set and recall triggered
    - Only pops from stack if it is empty
- Time how long it takes the user to complete:
    - Individual levels
    - The entire game
- Track number of fruits collected throughout the game
- UI
    - Persistent throughout levels
    - Displays:
        - Number of fruits collected (integer)
        - Total/global time (float to 2 decimal places)
        - Individual level time (float to 2 decimals places)
        - Time to beat each individual level (list of unrounded floats)
- Audio
    - Backgrounds music changes for different levels
    - Player audio
        - Jumping
        - Death
        - Recall point set
        - Recall triggered
- Pause menu
    - Buttons to:
        - Resume game
        - Quit game
        - Return to start screen
        - Skip level
    - Timers pause
    - Gameplay pauses

    - Background dims
- File for direct access contains:
    - Date of completion
    - Times for each level
    - Total time
    - Total fruits
    - Times sorted using merge sort

Level Design Objectives

- Enemies/traps
    - Kill the player
    - Can move on paths
    - Flying enemies can:
        - Fly between waypoints
        - Vary their height
        - Adjust velocity
    - Slime can be killed if jumped on, triggering a death animation
    - Spike turtles
        - Change between being a trap or not
        - Change between being normal ground or not
        - Spikes come in and out with animation
    - Falling spike head
        - Falls when player walks beneath it (trigger box)
        - Can adjust speed of falling
    - Rotating spike balls can have their rate of rotation and direction adjusted
    - Flame kills a player if it touches it
- Platforms
    - Swinging platforms
        - Can adjust speed of swing
        - Can adjust direction of swing
        - Can offset the swing of one platform from another
    - Moving platforms
        - Can be jumped on
        - Carry player with them
        - Can adjust movement speed
    - Disappearing/recurring platform
        - After a certain time become dim and cannot be stood on
        - Can change time for which platforms are active
        - Can change time for platforms to respawn
    - Rotating platforms
        - Platforms follow end chains
        - Can change speed and direction of rotation
    - On/off platforms

    - Change between active and passive by becoming dim or not dim and being able to be stood and jumped on or not
    - Can adjust time before changing state
    - Alternation can be offset between alternating platforms
- Flying item takes inputs from the player when the player jumps into it
- Finish flag loads next level after waiting x seconds
- Trampolines/fans launch player up
- Arrows show general direction of level progression

Have I met these Objectives?

Technical Objectives

Objective	Page Numbers	Explanation
Use the Unity game engine and C# programming language	Technical solution (136)	• Unity game engine used • All script in technical solution are written in C#
Implement object oriented programming, including classes and inheritance	Technical solution (136)	• Each script in technical solution uses its own class • All classes inherit from MonoBehaviour
Implement stack operations to allow player to set/recall their position	Recall script (173)	• Recall system uses 2 stacks for x and y coordinates respectively • Coordinates have same index in each stack • X and y coordinates are pushed onto stack to set recall point • X and y coordinates are popped and then player's coordinates are set to these values when recall triggered
Implement list operations for:	• Waypoint follower script (200)	• Waypoint follower script moves game

• Waypoint following • Times to beat levels • Quotes used if API import fails	• Timer script (191) • Import API script (156)	object towards first waypoint in the list (using a vector), when it reaches the waypoint it moves onto the next waypoint in the array, at the end of the list it moves towards the first game object in the array and repeats • List of times to beat levels 1-5 respectively is displayed in bottom left corner during gameplay, this list is also written to the text file and has a merge sort performed on it • Random element of 3 default quotes contained in defaultQuotes array used if API import fails
Implement file saving (text file and/or SQL) for direct access to save: • Date of completion • Times for each level • Total time • Times sorted using merge sort	Timer script (191)	• Data written to text file includes: ○ Date of completion ○ Times for each level ○ Total time ○ Timers sorted using merge sort • SQL not used because text file provides the functionality needed and being easier to implement
Utilise a merge sort to sort level times	Timer script (191)	• Merge sort sorts level times from smallest to largest • Uses two functions: ○ MergeSort ○ Merge
Utilise a recursive algorithm	Timer script (191)	MergeSort function calls itself

in the merge sort		until the list is sorted in its entirety, works by sorting smaller arrays and continually merging them
Import a web service API and use JSON parsing to give the player a message for completing the game	Import API script (156)	• Quote is imported from an API and is then displayed on the end screen • JSON parsing gets the correct part of the quote and author • Quote and author are concatenated to one string to be displayed

Use some form of complex maths	• Swing script (188)
	• Vectors:
	  ○ Player movement script (168)
	  ○ Waypoint follower script (200)
	  ○ Floating enemy script (151)
	  ○ Falling spike head script (146)
		• Swing script has complex rotation
		• Vectors used for movement of:
		  ○ Player movement
		  ○ Waypoint following (platforms, traps, enemies etc)
		  ○ Floating enemies
		  ○ Falling spike head

Use vectors for
• Player movement
• Platform movement
• Trap movement
• Enemy movement
	• Swing script (188)
	• Vectors:
	  ○ Player movement script (168)
	  ○ Waypoint follower script (200)
	  ○ Floating enemy script (151)
	  ○ Falling spike head script (146)
		• Vectors used for movement of:
		  ○ Player movement
		  ○ Waypoint following (platforms, traps, enemies etc)
		  ○ Floating enemies
		  ○ Falling spike head

Animate game objects
• Player has movement states for:
	• Player movement script (168)
	• Player animation state
		• Player animations:
		  ○ Running
		  ○ Jumping

○ Running ○ Jumping ○ Falling ○ Idle ○ Death • Basic animations for: ○ Enemies (spike turtle more advanced) ○ Traps ○ Platforms ○ Power ups	diagram (80) • Spike turtle script (180) • Traps/platform design (105) • Flying item script (153)	○ Falling ○ Idle ○ Death • Spike turtles alternate between spikes in or out • All enemies have basic animations • Platforms have basic animations • Flying Item has basic animations

Gameplay Objectives

Objective	**Page Numbers**	**Explanation**
Game meets requirements of client	• Client test (248) • Client (266)	• The client is happy that the game meets their requirements • Only client request not met was falling ceiling, but my client is happy that its replacement with the falling spike head is adequate
Camera follows the player's position	Camera controller script (136)	Camera x and y position is always the same as the player's position, x position is slightly different to the player so the player can still be seen
16 bit art style, similar to that of Super Mario World	• Game art and music (36) • User interface (36) • Gameplay screenshots (204)	• Art from the Pixel Adventure asset packs are in a 16 bit art style • "Pixels per unit" value set to 16 in the editor
Start screen	• User Interface (36) • Gameplay screenshots (204)	Start screen contains: • Game name • Game art • Buttons to start the game or quit the

		game
Main levels • Multiple levels • Varying length • Horizontal vertical scrolling • Levels in approximate difficulty order	Gameplay screenshots (204)	• 5 levels in total • Levels 1, 2, 3 are medium length • Level 4 is relatively long compared to levels 1, 2, and 3 • Level 5 is the longest level and usually takes a few attempts
End screen with random quote from API	• User interface (36) • Import API script (156) • Gameplay screenshots (204)	End screen contains: • Game complete message • Thank you for playing message • Random quote from API import (or a random default quote if API import fails) • Buttons to restart game from level 1

Allows keyboard, mouse and controller inputs	• Inputs (42)
	  ○ Input devices (42)
	  ○ Unity input system (43)
	• Player movement script (168)
		• Keyboard and controller can both be used to controller player:
		  ○ Movement
		  ○ Jump
		  ○ Pause
		  ○ Set recall point
		  ○ Trigger recall
		• Menus all navigated with a mouse

Recall system • Spawn point can be set (pushed onto stacks) • Player can teleport to the last spawn point (pop from stack) • Audio plays on spawn set and recall triggered • Only pops from stack if it is empty	Recall script (173)	• Spawn point can be set (pushed onto stacks) • Player can teleport to the last spawn point (pop from stack) • Audio plays on spawn set and recall triggered • Only pops from stack if it is empty
Time how long it takes the	• Timer script (191)	• Local timer shows

user to complete: • Individual levels • The entire game	• User interface (36) • Gameplay and text file screenshots (204)	time for current level • Global timer shows time since the start of level 1 • Both displayed in top right of screen • List of times for levels 1-5 respectively displayed in the bottom left
Track number of fruits collected throughout the game	• Item collector script (158) • Collectables manager script (138) • User interface (36) • Gameplay screenshots (204)	Total fruits collected during gameplay displayed in top right
UI • Persistent throughout levels • Displays: ○ Number of fruits collected (integer) ○ Total/global time (float to 2 decimal places) ○ Individual level time (float to 2 decimals places) ○ Time to beat each individual level (list of unrounded floats)	• User interface (36) • Gameplay screenshots (204) • Persistent UI script (164) • Timer script (191) • Collectables manager script (138)	• UI continuous throughout levels (persistent) • Displays: ○ Number of fruits collected (integer) in top left ○ Total/global time (float to 2 decimal places) in top right ○ Individual level time (float to 2 decimals places) top right ○ Time to beat each individual level (list of unrounded floats) in bottom left
Audio • Backgrounds music changes for different	• Video test (236) • Player movement script (168)	• Each level has background music (some levels have the

levels
• Player audio
  ○ Jumping
  ○ Death
  ○ Recall point set
  ○ Recall triggered
	• Player life script (166)
	• Recall script (173)
	• Trampoline script (198)
		same background music)
		• Player audio for
		  ○ Jumping (includes slimes and trampolines)
		  ○ Death
		  ○ Recall point set
		  ○ Recall triggered

Pause menu
• Buttons to:
  ○ Resume game
  ○ Quit game
  ○ Return to start screen
  ○ Skip level
• Timers pause
• Gameplay pauses
• Background dims
	• Pause menu script (162)
	• Gameplay screenshots (204)
		• Buttons to:
		  ○ Resume game
		  ○ Quit game
		  ○ Return to start screen
		  ○ Skip level
		• Timers pause by making time scale 0f
		• Gameplay paused by making time scale 0f
		• Background dims

File for direct access contains:
• Date of completion
• Times for each level
• Total time
• Total fruits
• Times sorted using merge sort
	• Text file design (125)
	• Text file screenshots (213)
	• Timer script (191)
		• Contains
		  ○ Date of completion
		  ○ Times for each level
		  ○ Total time
		  ○ Total fruits
		  ○ Times sorted using merge sort
		• File written with SaveTimesToFile() subroutine
		• Text file saved to the desktop

Level Design Objectives

Enemies/Traps

Objective	Page Numbers	Explanation

Kill the player	• Player life script (166)
	• Trap design (105)
		• When the player collides with anything marked with “Traps” it dies
		• Death shown by
		  ○ Death animation
		  ○ Death sound effect
		  ○ Level restarts

Can move on paths	• Waypoint follower script (200) • Trap design (105)	• Traps move along waypoints in list populated in the editor • E.g. saws
Flying enemies can: • Fly between waypoints • Vary their height • Adjust velocity	• Flying enemy design (106) • Floating enemy script (151) • Waypoint follower script (200)	Flying enemies can: • Fly between waypoints in list populated in the editor • Vary their height using vectors • Adjust velocity using vectors

Slime can be killed if jumped on, triggering a death animation	• Enemy death script (143)
	• Slime design (109)
		• Slime kills player if it hits its box collider for collision
		• Trigger box on top of slime bounces player and kills the slime
		• Slime death shown by:
		  ○ Death animation
		  ○ Slime disappears after animation finishes

Spike turtles • Change between	• Spike turtle script (180)	Spike turtles • Change between

being a trap or not • Change between being normal ground or not • Spikes come in and out with animation	• Gameplay screenshots (204)	being a trap or not by alternating tag between “Traps” and default tag • Change between being normal ground or not by alternating layer between “Ground” and default layer • Spikes come in and out with animation, repeats
Falling spike head • Falls when player walks beneath it (trigger box) • Can adjust speed of falling	• Falling spike head script (146) • Gameplay screenshots (204)	Falling spike head • Falls when player walks beneath it (trigger box) using a vector and rigidbody 2D • Can adjust speed of falling in the editor using serialize field
Rotating spike balls can have their rate of rotation and direction adjusted	• Rotate script (178) • Gameplay screenshots (204) • Rotating spike ball design (112)	• Spike balls kill player • Rate of rotation can be changed in the editor (negative speed to change direction)
Flame kills a player if it touches it	• Fire design (115) • Gameplay screenshots (204)	• Flame kills player • Base of flame acts as normal ground • Player dies if they touch the flame

Platforms

Objective	Page Number	Explanation
Swinging platforms • Can adjust speed of swing • Can adjust direction of swing • Can offset the swing of one platform from another	• Swing script (188) • Swing design (121)	Values can be adjusted in the editor: • Speed of swing • Direction of swing • Offset the swing of one platform from another • Time swing stops between rotations
Moving platforms • Can be jumped on • Carry player with them • Can adjust movement speed	• Waypoint follower script (200) • Moving platform design (121) • Sticky platform script (186)	Moving platforms • Can be jumped on because they are marked as ground • Carry player with them using sticky platform script • Can adjust movement speed in editor
Disappearing/recurring platform • After a certain time become dim and cannot be stood on • Can change time for which platforms are active • Can change time for platforms to respawn	• Recurring platform script (176) • Disappearing platform script (140) • Gameplay screenshots (204)	• After a certain time become dim and cannot be stood on • Can change time for which platforms are active in the editor • Can change time for platforms to respawn in the editor • Disappearing platforms replaced by recurring platforms during development do the game can always be completed
Rotating platforms • Platforms follow end chains • Can change speed and direction of rotation	• Rotating platform design (122) • Gameplay screenshots (204) • Waypoint follower script (200) • Rotate script (178)	• Platforms follow end chains using waypoint follower script (end chain is only waypoint) • Can change speed and direction of rotation in the editor

On/off platforms • Change between active and passive by becoming dim or not dim and being able to be stood and jumped on or not • Can adjust time before changing state • Alternation can be offset between alternating platforms	• On/Off platform script (160) • On/off platform design (124) • Gameplay screenshots (204)	• Change between active and passive by becoming dim or not dim and being able to be stood and jumped on or not (alternates whether collider is active) • Can adjust time before changing state in editor • Alternation can be offset between alternating platforms in editor

Other

Objective	**Page Numbers**	**Explanation**
Flying item takes inputs from the player when the player jumps into it	• Flying item script (153) • Flying item design (117) • Gameplay screenshots (204)	• Player game object becomes a child of the flying item game object when in the trigger zone inside the flying item so inputs affect flying item and player • Player behind flying item in z plane so player disappears into flying item
Finish flag loads next level after waiting x seconds	Finish script (148)	Finish flag loads next level after waiting x seconds after the player enters the finish trigger box
Trampolines/fans launch player up	• Trampoline script (198) • Trampoline/fan design (118)	• Player gets launched up with a vector when entering the trampoline/fan's trigger box
Arrows show general direction of level progression	• Arrow design (117) • Gameplay screenshots (204)	• Arrows show general direction of level progression • Arrows have no

		collision so don't interfere with movement of player, traps and platforms

Objective Reflection

I am happy that I have hit the vast majority of objectives. I have hit all gameplay and level design objectives and almost every technical objective. Initially, I thought that I would have missed off a few more objectives due to time constraints and the difficulty of certain features to implement. However, I have managed my time effectively so I have not been under pressure to cut out features. I have also used a range of resources such as the official Unity documentation to add more complex features, such as stacks for the recall system, the API import on the end screen and a range of trap/platform design. My objectives ranged from being very easy and quick to complete (such as object-oriented programming, traps killing the player and an individual level timer) to more aspirational, complex and time consuming objectives (such as the recall system, API import and file saving). I am happy that I pushed myself to complete the more aspirational objectives. I feel this has improved my programming skills, research skills and problem solving skills.

However, I have missed out SQL on the following objective:

- Implement file saving (text file and/or SQL) for direct access to save:
    - Date of completion
    - Times for each level
    - Total time
    - Times sorted using merge sort

Whilst I have technically met the objective here - because I have used a text file - I did not implement SQL in the final build of the game. The reason I chose to not pursue the implementation of SQL any further is because I was happy that the text file implementation was easy for the end user to interact with and to implement. Adding SQL in addition to the text file does not give my client much more functionality, the only read advantage of SQL over the text file would be ordering times, searching through times and a neater display of information. My client hasn't informed me that file saving is of a high priority, so I decided that my development time would be better used developing other game mechanics such as level design and the recall system. I feel that if I had put more time into SQL, it would have put me under pressure to rush other more important parts of the system which would not be good for my client.

Overall, I am very happy that my game is complete, given the scope of this project. I feel I have hit all of the objectives that are needed. The only part of an objective I missed was SQL, and I am satisfied that the text file implementation instead still leaves the game feeling complete.

Client

Client Test Result

Client Request	Completed?	Possible improvements
Similar to Super Mario World and other older 2D platformers	Yes • The game is focused on tight platforming sequences of increasing difficulty • Tight controls • Both have a 16-bit art style	• Add boss fights (e.g fighting Bowser in Super Mario World). • Add unlockable movement features
Continuous side scrolling	Yes • Each level scrolls from left to right • Camera follows player's movement • Some levels also have a large vertical element • Camera is never locked, it always scrolls in line with the player	Force the camera to always move independently of the player's movement to increase time pressure.
Moving platforms	Yes • Moving platforms move between waypoints in a list • After completing the list of waypoints, the platform moves back to the start and repeats • Variations such as rotating platform	None
Time element	Yes • Global timer to show time to beat all levels • Local timer to show time to beat individual level • Array of times to see	Have a timer count down from x seconds and if the player runs out of time they have to restart.

	time to beat each level individually • Times saved to text file for direct access • Merge sort sorts times from shortest to longest	
Cloud item (from Super Mario World)	Yes • The flying item replicates the behaviour of the clouds in Super Mario World	None
Disappearing/falling platforms	Yes • Recurring platforms disappear after being stepped on for x seconds, and return after y seconds	Could get platforms to fall after a certain time, but the platforms returning is important to make sure levels are always completable.
Swinging platforms/pendulums	Yes • Swings rotate for 180 degrees • Can adjust direction and speed of swings in editor • Can offset swing patterns between platforms	Make swing carry player.
Falling ceilings	No • Made redundant by the falling spike head • Most levels don't feature a roof	Add falling ceilings.
Checkpoints	Yes • No physical set checkpoints • Player can set their own checkpoints using the recall system	None
16-bit art style	Yes • The assets imported from Pixel Adventure 1 and 2 are in a 16-bit	None

	art style	
Few, longer levels	Yes • Most levels are long, but still very in length • Level 3 is fairly short to add variety • Level 5 is very long to be a final challenge	Add more levels.
Non-rhythm platformer	Yes • Each level has normal background music • Platforming is not based on the music timing	None

Client Feedback

What features of the game are you most happy with?

Simplicity, balanced difficulty, straightforward controls, retro 90s feel of the game.

Are there any features you feel are missing from the game that you wanted to be there?

No

Do you feel the game has met your requirements?

Yes and surpassed them. The inclusion of “falling spike head” instead of falling ceilings works well. The recall system works as a checkpoint system. The use of a text file instead of an SQL database is more user friendly.

Do you feel the game is complete?

Yes

If I could expand the game with more development time, what else would you like to see added?

- More power ups and items (only flying item/cloud) currently. Boss fights at the end of certain levels/the whole game.
- Forced camera scrolling to add time pressure on certain levels. Have timer count down on certain levels. Having a few more levels generally.

Reflection On Client Test and Feedback

I am very pleased to see that my client not only feels my game is complete, but has surpassed the requirements that they are looking for in a 2D platformer. I knew that my client needed the core of this game to be the main gameplay - particularly the 16-bit art style, tight/simple controls and level difficulty. Therefore, I always kept this in mind when developing the game.

I have met the vast majority of my client's requirements completely. The core requirements to the game were the 16-bit art style, tight/simple controls and level difficulty. I have met each of these and my client was very satisfied with these aspects in the actual test. Particularly, my client liked the 16-bit art style from the Pixel Adventure asset packs I used. They liked how similar the art style was to the art in Super Mario World and the clarity of different game objects. The client liked the controls on the controller in particular and also recognised the convenience of also allowing for keyboard input. My client also thought the level difficulty was very good, particularly compared to more modern 2D platformers which tend to be easier and have more assists to help lower skilled players. Hitting these core objectives from the start were instrumental to making sure my client was happy with the final system.

I also hit more objectives outside the core gameplay element. The client liked that you could see the essential time information in the corners of the screen, but still ensuring it doesn't interfere with the core gameplay. This adds an element of time pressure as my client requested in the analysis stage. They liked the text file implementation because it shows important information about each playthrough and is easy to open, understand and interact with compared to other file types, such as an SQL database. They liked all of the platforms - both the ones suggested during analysis and the ones I added - which helped meet their requirement of increasing difficulty of levels; the client also liked the enemies I added for the same reason. My client requested a checkpoint system. They were very satisfied with the unique recall system I created using a stack rather than a traditional recall system. They were particularly happy that the new recall system did not distract from the main gameplay, but could be accessed simply using the relevant buttons. Also, I made sure that the background music was not correlated with the actual platforming, as my client did not want a rhythm based platformer. The tempo etc of songs follow along with the general theme and difficulty of each level.

My client did suggest some additional features they would be interested in should I take the project beyond the scope set originally. Their recommendations were more power ups, boss fights, forced camera scrolling, timer counting down and more levels in general. More power ups could be a great addition so long as they aren't too confusing for the player. I would want to make sure that the power ups benefit a particular level and are not just there for the purpose of adding power ups. An example of this could be an item that reduces gravity (this would likely need a specific level design) or an item to unlock a new movement feature permanently. Boss fights would be very time consuming to add, but a feature I would be very interested in to change the style of gameplay and expand on the intensity of gameplay developed in later levels. I wouldn't utilise forced camera scrolling in many levels, but I could see its place in a shorter, fast paced level. I would also apply this to a timer that counts down because a long countdown

won’t provide much time pressure; however a timer which is shorter and matched to a specific level would make for very intense gameplay. More levels in general would be easy to add because of the ‘prefab’ game objects I have saved, but would more importantly allow me the space to add the features mentioned above whilst making sure levels are not crammed too much with platforms, enemies and power ups.

However, I did not meet the client requirement of a falling ceiling. I felt that this was made redundant by the “falling spike head” enemy, and thus my development time would be better spent on other features. Later features I added include the recall system and API import. This also gave me more time for testing and fixing bugs. I am happy that my game is more complete with these features than they would be if I prioritised the falling ceilings. My client agrees that the ceilings are obsolete because of the “falling spike head”. The client feels that the game is complete. Therefore the client and I are both happy that I made the right choice in not including falling ceilings in the final system.

In conclusion, I am very happy that my client is happy that my game is complete and surpasses their expectations. To achieve this, I made sure that the core gameplay was perfected, as this was highlighted by my client as the most important part of the game during the analysis stage. By doing this early on, along with not adding redundant features, I had extra time towards the end of development to test the game and add features beyond what my client needed. This left the client impressed with the completeness of the final game and additional features that I added.

Overall Evaluation and Reflection

What went well?

Overall, I am satisfied I have hit the vast majority of my objectives and client requirements set out in the analysis stage. Both my client and I agree that the game is complete within the scope of the project and beyond. I will explain how I achieved this in development.

To start off, I focused on the core gameplay elements such as the 16-bit art style, tight/simple controls and level difficulty. My client was very clear that this was the most important aspect of the game. Therefore, I made the objectives that correspond to these requirements high priority and hit these early on during development. I chose the Pixel Adventure asset packs before I started development as I was happy that my client would be very happy with the art style, because it is similar to that of Super Mario World. I wrote the player movement script early on in development and played around with the values for movement speed and jump height to make sure that the controls were simple and tight as requested. I then ensured the level design complimented this movement to form difficult platforming sections. I also made sure that the levels had a difficulty gradient by designing them all in a fairly short time period so I remembered how difficult the next level needed to be. I made any level difficulty changes based on whole game tests to make sure each level is generally more difficult than the last. Making the

core gameplay elements a priority and taking care of this at the start of development made sure I did not have to rush such important things at the end. My client was very satisfied with the core gameplay elements in their feedback which I analysed earlier.

I managed my time effectively to ensure I was always spending development time efficiently. Before I started off development, I chose to use free assets from the Pixel Adventure asset packs instead of creating my own art. My art skills are not the best, so creating sprites and corresponding animations would not only make the game look bad (which my client would not be happy with) but also waste lots of time before I have even started writing actual code and building levels. I then prioritised core gameplay elements and their corresponding objectives because my client emphasised the importance of this. By sorting this at the start of development and not including redundant features such as falling ceilings and SQL, I had more time for the non-core gameplay elements and features and later stages of development.

I had time left to allow for lots of testing, including 5 build tests. This testing allowed me to fix lots of bugs. There were some larger bugs such as issues with the timer which would have made the game feel very incomplete and broken which my client and I would not have been happy with. These were fixed to make sure the final game was very much playable. I also enjoyed the challenge of fixing all of the smaller bugs. Whilst these wouldn't always be noticed during gameplay, I felt fixing these are important to make the user feel the game is entirely complete and has lots of attention to detail. As a result of these iterative tests, I am glad that my client did not find any bugs when they tested the final build of the game and feel that the game is complete.

I also had time left to hit some more aspirational objectives towards the end of development. The recall system I designed using stacks was ambitious, but because I left enough time to solve any issues with it, I managed to complete the feature and move onto the next feature. I also imported an API and parsed JSON to give the user a random message at the end of the game. I left this to the end because it is not very important and is more of a finishing touch to the game. I again managed to complete this feature to be used in the final game because I left enough time. It is smaller features like this that make my client feel I have surpassed their requirements.

Any objectives I set and requirements from my client that were not met have been dealt with. I did not hit the SQL part of the file saving objective. I always intended to only include either text files or SQL but not both. My client preferred text files over SQL because they were more simple and user friendly. I also think text files are easier to implement technically than SQL databases. I did not meet my client's request of falling ceilings. However, they are happy that the inclusion of the "falling spike head" is more than sufficient to replace the falling ceilings. By not spending time on redundant features such as these left time for more important features towards the end of development as discussed above which would have satisfied my client much more than making sure every single objective and requirement is met perfectly.

How could the project be improved or expanded?

Whilst my client and I are both happy that the game is complete given the scope of the project and beyond, I did not hit all objectives and client requirements in their entirety. Specifically, I did not hit the SQL part of the file saving objective. I always intended to only include either text files or SQL but not both. My client preferred text files over SQL because they were more simple and user friendly. I also think text files are easier to implement technically than SQL databases. Therefore, to improve the project I could also add an SQL database to store and compare level times etc. The advantage of SQL over text files is being able to data by certain columns e.g. time and alphabet. Having both SQL and text files would allow the user to use whatever file type they prefer. I also did not meet my client's request of falling ceilings. However, they are happy that the inclusion of the "falling spike head" is more than sufficient to replace the falling ceilings. By not spending time on redundant features such as these left time for more important features towards the end of development as discussed above which would have satisfied my client much more than making sure every single objective and requirement is met perfectly. Given more development time, I would add falling ceilings in new levels to add some variety and not just use the falling spike head because more levels with the same traps would get repetitive.

Additionally, my client did suggest some additional features they would be interested in should I take the project beyond the scope set originally. Their recommendations were more power ups, boss fights, forced camera scrolling, timer counting down and more levels in general. More power ups could be a great addition so long as they aren't too confusing for the player. I would want to make sure that the power ups benefit a particular level and are not just there for the purpose of adding power ups. An example of this could be an item that reduces gravity (this would likely need a specific level design) or an item to unlock a new movement feature permanently. Boss fights would be very time consuming to add, but a feature I would be very interested in to change the style of gameplay and expand on the intensity of gameplay developed in later levels. I wouldn't utilise forced camera scrolling in many levels but I could see its place in a shorter, fast paced level. I would also apply this to a timer that counts down because a long countdown won't provide much time pressure; however a timer which is shorter and matched to a specific level would make for very intense gameplay. More levels in general would be easy to add because of the prefab game objects I have saved, but would more importantly allow me the space to add the features mentioned above whilst making sure levels are not crammed too much with platforms, enemies and power ups.

There are some more features not mentioned by my client that I would also be interested in adding. During development, I did consider each level having its own movement feature or unlocking movement features as the game progresses, similar to the system in Rayman Origins. However, given the development time, I would have struggled to implement many movement features at a high enough standard to include in the final game and had to leave out other features from later development such as the stack recall system and API import. Furthermore, with only 5 levels, I would feel that each movement feature would feel like a novelty because there is not enough playing time to truly master a movement feature before unlocking the next one. Given more production time, I would develop this idea further with more levels to work with.

Some movement features I could add include sprinting, dashing and gliding. I also considered adding a multiplayer feature (offline or online). I did not explore this because my client did not want multiplayer, but it could definitely be added if I had more development time. This would let players compete for who can beat the level fastest or add a cooperative problem solving element similar to that of Fireboy and Watergirl. Online multiplayer in particular would be a great challenge to implement but I believe it would be rewarding to do so. Lastly, I would maybe add some music levels. Whilst I would not want too many, they offer a nice change of pace to the game. I would go for a similar rhythm to non-rhythm level ratio to that seen in Rayman Legends, where there is one music level per world of approximately 10 non-rhythm levels. I would enjoy the iterative testing required to match the platforming to the music. However, getting music produced would be costly because I have no experience with making music.

Conclusion

Overall, the most important factor is that both my client and I are happy that the game is complete and even surpassed requirements set out by my client. Therefore I consider the project to be a large success and will pursue even larger scale projects in the future and may even expand this project with a wider scope. Whilst I have not hit every single objective and client requirement in its entirety, I have made sure that my client is satisfied with the replacement feature as discussed above and the use of text files over SQL has its own advantages and is the preferred file type by my client. Therefore, by making sure my client is happy with these small details, I added extra features to expand the game beyond their expectations - such as the quote imported using an API - and make sure I fixed all the bugs I found during testing. I also saved time by choosing to use premade assets instead of making my own which would have looked worse anyway. This use of my saved time allowed me to make my client more happy than making sure every small objective and requirement were hit so overall I believe I made the right choices during development. If I were to expand the game with a wider scope in the future, I would populate more levels with new power ups, boss fights and rhythm based sections. I would also add new features such as multiplayer, forced scrolling and a timer counting down to both new and existing levels. This project has taught me many new things; the most important being object-oriented programming, practical use of abstract data types such as stacks and how to import an API and parse JSON.