Table of Contents

Analysis

Introduction

My idea is a digital adaptation of a variant of Ultimate Noughts and Crosses previously made by me and a friend (dubbed ‘UTTTTT’). In traditional Ultimate Noughts and Crosses, nine small Noughts and Crosses boards exist in one large Noughts and Crosses board. The first player starts by playing anywhere in any of the small boards. The second player then has to play in the small board corresponding to the square that the first player played in. When a small board is won, the entire board is taken by the winner. The game ends when a player wins on the large board. Our variant of the game adds many additional features, the most notable being time travel, an in-game shop and quick-time events.

My project will make playing UTTTTT much easier and more accessible. Despite being a two-player game, three people are required to play. This third person is in charge of managing the extra mechanics the game has compared to regular Ultimate Noughts and Crosses. This third person is not always available and must have a good understanding of the game. Even with this, they may make errors. Because of this I believe a computational approach is suited to this project, as the computer will act as the third person managing the mechanics. This will make the game much easier and more enjoyable to play. This will additionally make playtesting easier, which will also make development of UTTTTT itself much easier.

For my project, I would like to include as many features as possible from the original game, however I am aware that I likely will be unable to. Because of this, I will need to conduct research to find out how the prioritise the features to implement. To do this, I plan to ask people who have helped to develop the original UTTTTT what they would like to see the most in a digital adaptation. The game will need to include some form of multiplayer to be a true adaptation, either remote, local or both. I will further research which of these options will best suit the project. At this stage, I plan for my project to be a desktop game and I am investigating using Unity to do so given the games complexity.

Stakeholders

As this project is intended to be a digital adaptation of UTTTTT, I am primarily targeting this project towards my friends that I play UTTTTT with. Therefore, I will be developing the features of this project with these people in mind first. I hope to include this group of people throughout the development of this project to ensure that it fits their needs as we will use this game to play with each other.

However, I am also interested in using my project to introduce more people I know personally to UTTTTT who may have previously been discouraged by its difficulty to play. To do so, I will ask friends what features of the game they would find enjoyable so that the game is more appealing to them.

I have conducted a survey to identify what features of the game would be ideal to include, detailed below.

Survey

Questions

1. **Have you played UTTTTT before or otherwise helped in its development?**

This question lets me know which respondents are familiar with UTTTTT. This is important as these are my initial stakeholders, so I can give their responses more weight when I am making decisions.

3. **How familiar are you with normal Ultimate Noughts and Crosses? (only shown if respondent answered 'No' to 1)**

I have used Microsoft Form's branching feature so that only people who responded 'no' to the previous question respond to this question.

These are the people I am interested in introducing the game to, so gauging their knowledge of the original game informs me of how much tutorial and explanation I should include in my project.

4. **What features would you find enjoyable if added to Ultimate Noughts and Crosses? (only shown if respondent answered 'No' to 1)**
2. **What features of UTTTTT would you find essential in a digital adaptation? (only shown if respondent answered 'Yes' to 1)**

Both questions serve similar purposes in telling me what features would be desirable to add to the game. I will prioritise the answers to the second question, as those respondents come from the people, I am specifically making this digital adaptation for. However, the first question lets me know what features would interest or disinterest those who I want to introduce the game to if I am to consider their preferences.

5. **Are there any additional features would you find enjoyable?**

The responses to this question will tell me what mechanics stakeholders would enjoy in the game that does not exist in the original UTTTTT. This will mainly allow me to find out how to make the game further appealing to those who haven't played it before. However, I will have to be very selective of the features suggested as I do not want to change the nature of UTTTTT much.

6. **How important would you find the ability for local multiplayer?**
7. **How important would you find the ability for remote multiplayer?**
8. **How important would you find the ability to play single-player against an AI?**

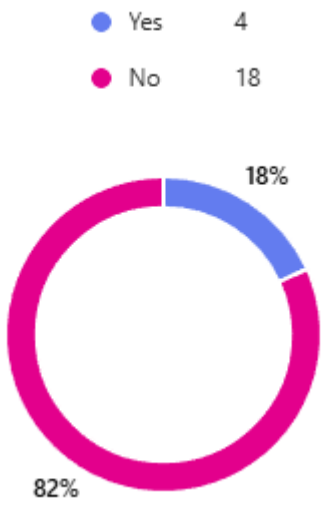
These questions will tell me what forms of multiplayer my stakeholders find more or less important. This will guide what forms of multiplayer I will include in the project.

9. **Would you prefer the game to be a desktop app, mobile app or website?**

This will tell me what format is most desired by stakeholders, or if they have a preference at all. With this information I can design my game in a way that will most satisfy its users.

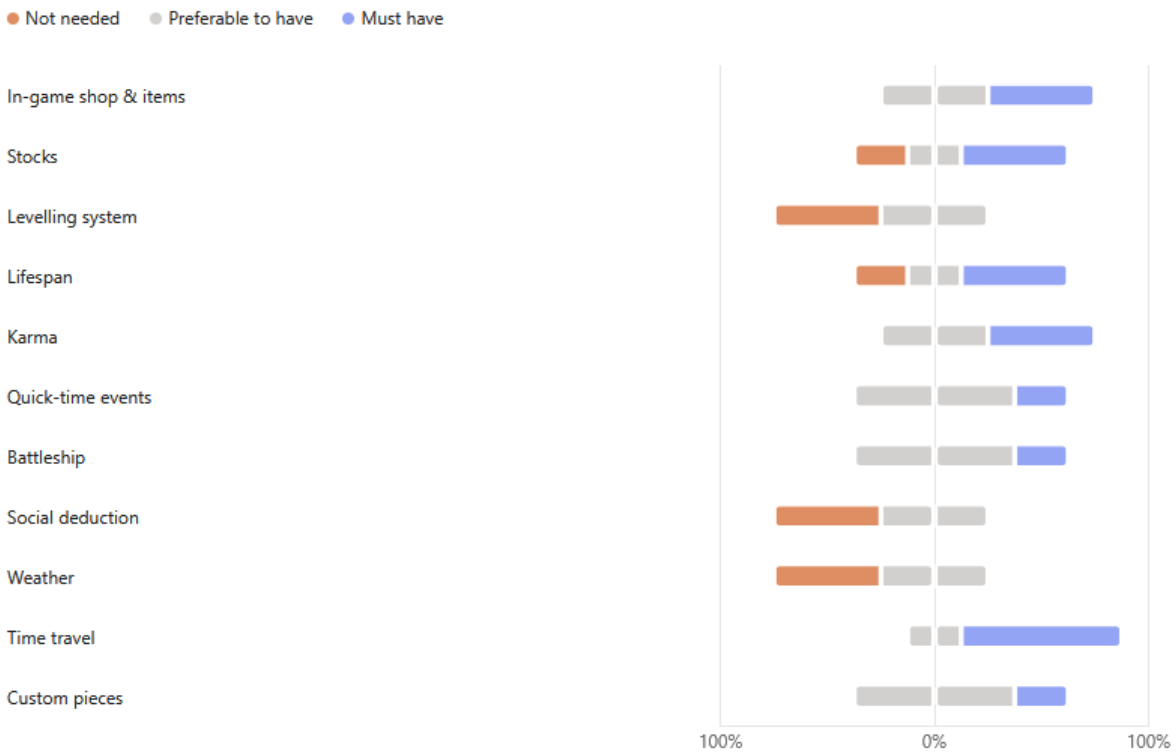
Answers

1. **Have you played UTTTTT before or otherwise helped in its development?**



The purpose of this question has been analysed above, and these results are expected as I purposefully gave this form to specific people.

2. **What features of UTTTTT would you find essential in a digital adaptation?**

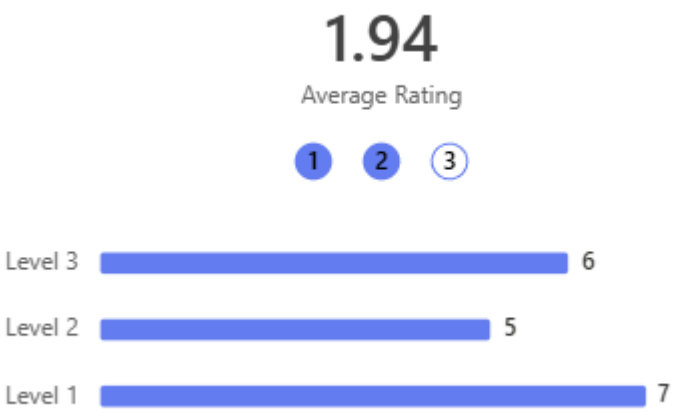


Results show that time travel is near universally considered a must-have, with the in-game shop and karma system also being considered important. Because of this, I will ensure that I include these features. This is to ensure that my project is a faithful recreation. Although quick-time events were considered less essential, it is important for the shop system as it is the main way of earning coins. Therefore, I will also ensure to include those.

Other features that were considered preferable to have were battleships and custom pieces. I will mark these as desirable and try to include them, but will prioritise them less than the features above as they are less necessary for my project to be a successful adaptation.

Levelling system, social deduction and weather were reported as not needed, while stocks and lifespan were controversial, so for now I will consider these optional features as they are not unanimously considered important by my main stakeholders. This means they are not essential for my project to be successful.

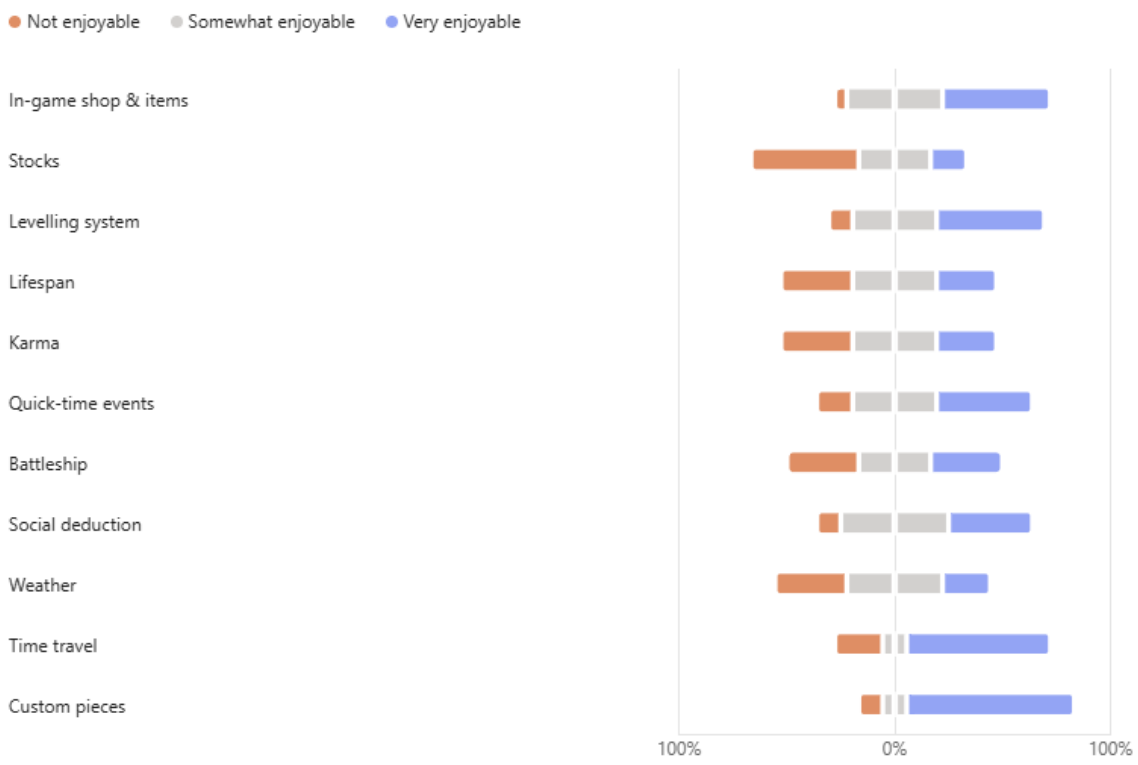
3. How familiar are you with normal Ultimate Noughts and Crosses?



Level 3 was labelled as “very” while level 1 was “not very”.

Levels of familiarity to the base game of Ultimate Noughts and Crosses are split almost evenly, with 6 experienced players, 7 unexperienced players and 5 in the middle. This shows that the addition of a tutorial for normal Ultimate Noughts and Crosses will be desirable as there are many unexperienced or somewhat unexperienced players. However, the tutorial should be optional, as there is a significant proportion of experienced players. This tutorial will help players understand how to play the game, which will allow me to more easily introduce UTTTTT to other people using my project, thus making my project more successful. However, because my main stakeholders will already know how to play the game and therefore have no need for a tutorial, this feature will be marked as desirable rather than essential.

4. **What features would you find enjoyable if added to Ultimate Noughts and Crosses?**



Results show that custom pieces and time travel would be very enjoyable for new players, along with the in-game shop and levelling system. Due to the results of question 2, I had already marked time-travel and the in-game shop as essential, so no change is necessary. However, I will add custom pieces to this list as it was already considered preferable to have in question 2 and it is overwhelmingly marked as very enjoyable in this question. The levelling system's priority will be raised slightly but not significantly as it was controversial in question 2. This will make my game more appealing to those I want to introduce it to without sacrificing the wishes of my main stakeholders.

Although social deduction was considered somewhat enjoyable and opinions on the karma system were mixed, I will not take these results into account. This is because they conflict with the results of question 2, whose results I am giving more weight. This is because they are my main stakeholders as people who are active players of UTTTTT. This ensures that the project remains first and foremost an adaptation of UTTTTT, while making it appealing to new players second.

5. Are there any additional features would you find enjoyable?

Responses
Scrabble, chess if it doesn't fall under "custom pieces", event cards
Can you add dragons in somehow please And gacha Mmm gambling
Seige mode
Battle mechanic
Upgrading individual pieces, Map Maker
Streak of wins
minigames
ga
No
gambling
More than 2 players in one game
meow

Event cards are an existing feature of UTTTTT that expands on the quick-time events feature. Quick-time events are minigames that occur between turns. Sometimes instead of a quick-time event triggering, an event card will trigger instead. Event cards cause a direct effect to the game, for example by causing a player to lose all their coins. I did not include it in question 2 or 4 of my survey because at the time, the feature had not finished development. However, it has now and as a main stakeholder has expressed a desire for it, I will be including it.

I will not implement scrabble, more than two players, chess, a siege mode, a battle mechanic, upgrading individual pieces and a map maker as I feel these features will change too much and therefore make the game too different from UTTTTT. I would like to avoid this as it would detract from the main purpose of my project as an adaptation of UTTTTT.

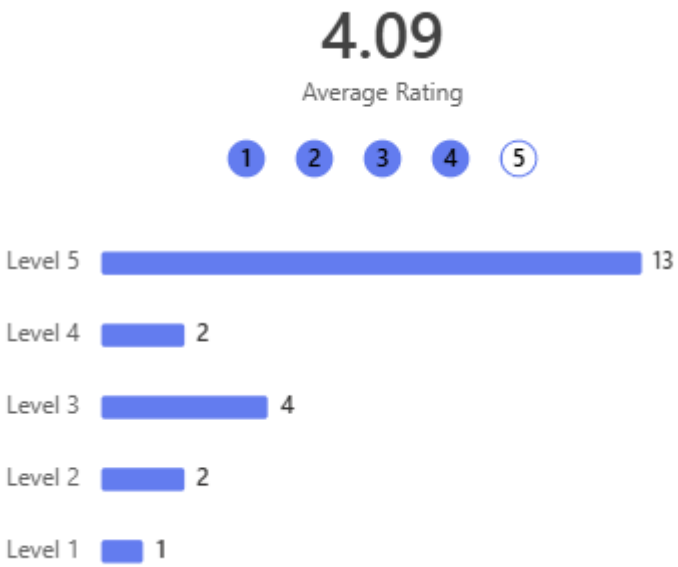
Minigames will be included as a form of quick-time event. Gacha/gambling can be added to the in-game shop as a purchase. Dragons could also be added to the in-game shop but may also appear in events. Meowing can be included as a meow button that when pressed, causes a meow sound to play. A win streak could be implemented alongside a rematch option. None of these features will be too difficult to implement and it will not change gameplay. This means that adding them will make these

respondents much more interested in the game while not taking away from my projects main goal, which is what I want and will my project more successful. Therefore, I will mark them as desirable to add.

6. **How important would you find the ability for local multiplayer?**

Analysis of results continued with question 7's below.

7. **How important would you find the ability for remote multiplayer?**



For this and the following two questions, level 1 was labelled as 'Unimportant' while level 5 was labelled 'Very important'.

In question 6, almost every respondent answered in levels 3, 4 and 5. This shows that local multiplayer is viewed as quite important although not overwhelmingly so. This means it would be ideal for me to add it.

In question 7, show that the majority of people view remote multiplayer as very important. These results were more unanimous and less mixed than the prior question, showing that there is more consensus on this issue.

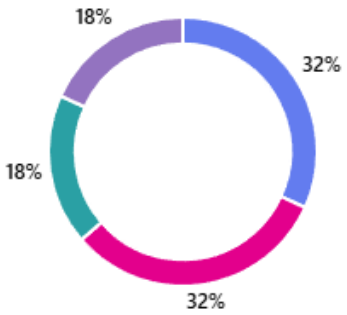
Both questions received similar results implying that both forms of multiplayer are desirable. Because of this, I will need to do more research before deciding on a form of multiplayer to implement or if it is possible to include both.

8. **How important would you find the ability to play single-player against an AI?**

Although the average rating is 3.36, which implies some importance, results are a lot more mixed than previous questions. Because of this, for now I will mark it as optional. It is also not essential to the success of my project, as the goal of my project is to facilitate playing UTTTTT specifically between two people. I will also need to investigate if it is a feature that is feasible for me to add in the time I have.

9. **Would you prefer the game to be a desktop app, mobile app or website?**

Desktop app	7
Phone app	7
Website	4
No preference	4

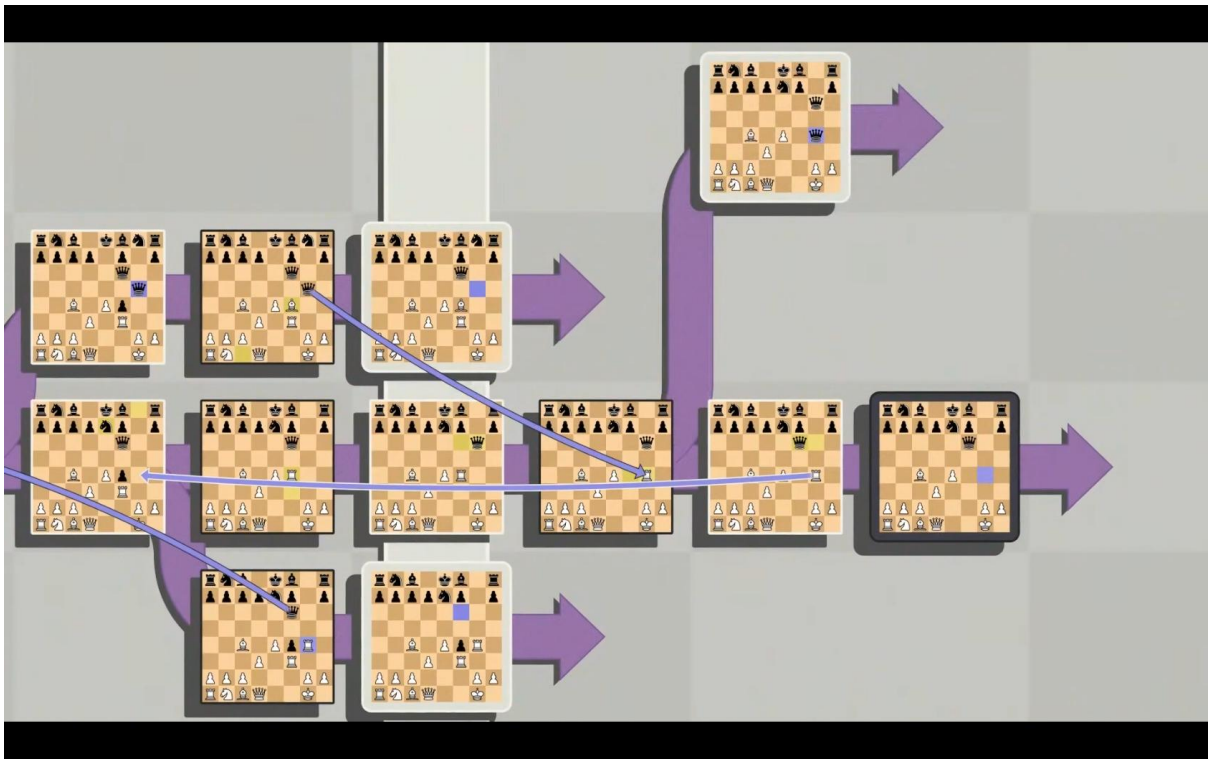


Most respondents expressed a preference, with results split between desktop and phone. Because of this, I will investigate making both a mobile and desktop version of my game.

Research

5D Chess With Multiverse Time Travel

5D Chess With Multiverse Time Travel was one of the largest inspirations for UTTTTT. It is a chess variant in which pieces can move through time and timelines. I am researching this game as an example of an board game adaptation and also as a game involving time travel.



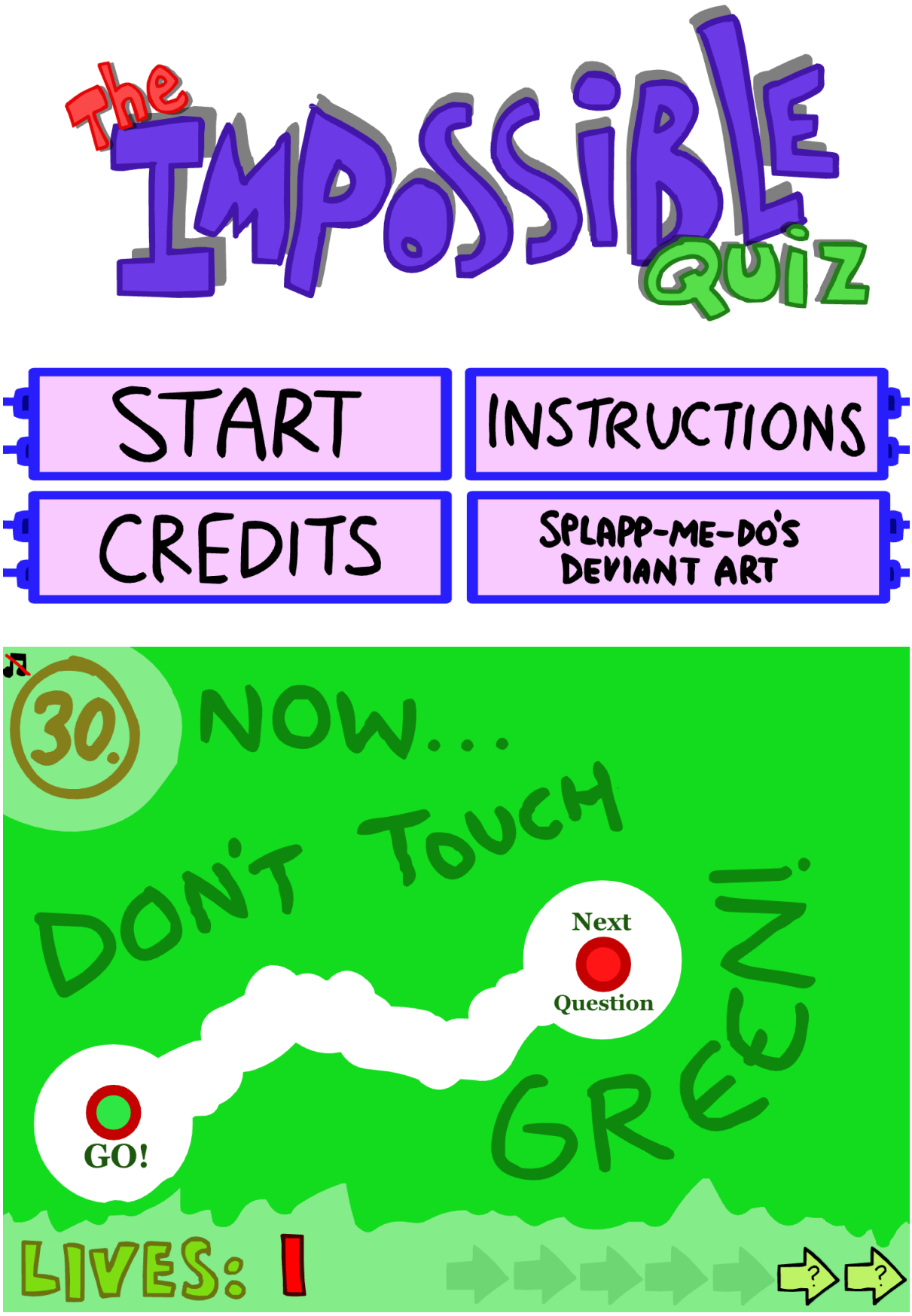
1 Screenshot taken from https://youtu.be/EBzX1ybigmw

Ideas to take forward	Ideas to drop
Simple graphics The graphics use few colours and are not very detailed which make them very clear and readable. This is very important as my game is quite complicated; simple graphics make the game easier to understand. **Clear timeline layout** Despite the complicated multiverse mechanics, it is easy to visually tell what is happening where and when. It is clear what boards belong to what timelines and where timelines branch off. Telling timelines apart is also easy due to the grid-like layout. The coloured outlines on	**Puzzles** My game will involve additional features besides time travel that are luck-based. This makes puzzles impossible to implement. **Formal graphics style** I find the font and style of the pieces used quite formal. As my game is intended to be purposefully silly and over-the-top, I believe this style would not suit it and would cause a tonal clash. **History & parallel view** In *5D Chess With Multiverse Time Travel*, it is possible to activate a history or

the board also allow for players to differentiate between turns in the past. This allows the user to strategies easily between timelines, which is important in a strategy game. Because UTTTTT similarly involves a lot of strategy, I will design my layout similarly. **Rules** In *5D Chess With Multiverse Time Travel*, there is a rules section explaining the rules of the game. I would like to include a similar rules section as my game similarly has complex mechanics that need explaining. A rules section would ensure that all players fully understand how to play the game. This is useful both for experienced players who may be familiar with the mechanics but not how they are implemented in my project and new players who are entirely unfamiliar with the mechanics. **Animated menu background** The menu features chess pieces falling towards the viewer, giving the impression of the chess pieces moving through space. I find this appealing as it is a nod to the features of the game and also adds a sense of excitement and intrigue to an otherwise plain main menu. I would like to do something similar as it helps to make the game more engaging.	parallel view. This changes the games perspective from top-down to slanted and also shows all possible moves of a given selected piece with arrows. This makes it easier to see how pieces move across or through timelines given their rules. However, this will not be necessary in my project as there are no rules to where you can move pieces across time as you can put them anywhere. Adding arrows would cause clutter and confusion.

The Impossible Quiz

The Impossible Quiz is a point-and-click quiz game composed of 110 questions, intended to be extremely difficult. I am researching this game as an example of a game intended to be 'silly' and also as a game that mimics how quick-time events will be implemented.



Ideas to take forward	Ideas to drop

Range of questions
The quiz mixes typical multiple-choice questions with riddles, puzzles and more interactive minigame-like questions. Having a mix will make events feel more unpredictable and which will make them more exciting and engaging. I will ensure that I include a similar mix of events from the original UTTTTT.

Skips
Having a finite number of skips will allow for increased strategy as users will have to consider the consequences of using up a skip. Skips can also be added to the in-game shop feature, giving that mechanic more depth and making sure different aspects of the game flow well together.

Hand-drawn graphics style
I believe hand-drawn nature of the graphics fit my project well, as it gives the game a more casual feel. This fits my project well as it is primarily being made for my friends and people I know personally.

Question timer
In *The Impossible Quiz*, for timed questions a bomb is displayed in the top right corner. Inside is a countdown, which turns red at 5. This induces a sense of urgency and panic within the player, which is equally fun and engaging. I would like to implement a similar countdown in my game.

Style of humour
The quiz features some references and jokes that I find outdated and don't find appealing. Events in UTTTTT derive humour more from how ridiculous and unexpected they are, therefore if I am to make new events I will not incorporate this style of humour.

Trick questions
This works for the Impossible Quiz as a game intended to be frustrating. UTTTTT is not intended to be frustrating in this way so if I am to make new events I don't believe unfair questions would be a good idea.

Lives system
Receiving a game over suits *The Impossible Quiz*, however quick-time events are not a lose condition in UTTTTT. Making it one in my adaptation would not be a good idea as it would overplay the importance of quick-time events and overshadow the rest of the game. Because of this, I would rather replace this with a system that prioritises rewarding success rather than punishing failure. Players instead earn coins which is what happens in UTTTTT and suits the game better.

Untimed questions
Most of the questions in the Impossible Quiz are untimed. Although quick-time events are usually untimed in the original UTTTTT, this is due to technical limitations and ideally they would be timed. This is because the lack of a timer leads to players spending too long on quick-time events, which are meant to be completed in a short amount of time. This greatly increases the time it takes to complete a game to an unenjoyable degree. Therefore, I will take this opportunity to add timers to quick-time events.

Turmoil

Turmoil is a simulation game based around becoming a successful oil entrepreneur. I am researching this game primarily as an example of a game with an in-game shop.

Ideas to take forward	Ideas to drop
Shop display The items available for purchase as well as the player's current balance are displayed at the top of the screen, not in	**Dollar currency** Unlike Turmoil, UTTTTT is not set in a specific time period or location so it does not make sense to have a real-world

a separate menu. I find this important for my game as it allows players to make an informed decision on what to buy as they can see both the current state of the game and all the items available for purchase. Each item is also displayed visually with a simple icon. This makes it easy to tell at a glance what items are available, so players can make quicker decisions.

Buying with keyboard shortcuts
Items can be bought by clicking on their icons or buy pressing the key linked to that item. I would like to add this to my game as it will allow experienced players to access the items they want quicker.

Item descriptions when hovered over
This allows newer players who may be less familiar with the game to find out what each item does without cluttering the screen for more experienced players by having the description always present.

Tutorial
Turmoil features a short skippable tutorial level that showcases the basics of the game. In this tutorial, a character called the mayor introduces features one by one with a small text pop-up explaining them. He guides the player through completing the level. This is an effective way to teach the player how to use the UI and also how to complete levels. I would like to implement a similar walkthrough tutorial to ensure inexperienced players understand the flow of the game and the UI.

currency. I will likely instead implement a more vague idea of currency such as “coins” which is how the currency system is usually implemented in UTTTTT.

Characters & campaign
Turmoil is a single-player game, therefore AI controlled NPCs are needed to play against. I intend my project to be a multiplayer game, therefore adding characters to play against would be unnecessary. As my game will not have characters, this will make a campaign mode also unnecessary. However, I may include an option to play single-player against an AI.

Requirements

User requirements

E – essential
D – desriable
O – optional

Feature				Explanation	Justification	Importance
		Sub-feature				
1	Main menu	1.1	Title	A display of the name of the game.	This is included within each game I researched. I believe it is an essential feature as it makes it clear what game the user is playing and adds immersion.	E
		1.2	Play button(s)	A button for each mode of playing which when pressed, starts the game in that mode.	This is another feature in all the games I researched. It is important as it will allow a player to select what mode they want to play in. It also ensures that the player is ready to play the game as they can decide when they start.	E
		1.3	Tutorial	There will be a tutorial button on the main menu which will open a tutorial. The tutorial will be a walkthrough of a game and will have two parts, the first of which is skippable. The first part demonstrates how to play normal Ultimate Noughts and Crosses while the second part explains each added mechanic. There will also be short, text pop-ups to aid with understanding.	With a tutorial, players can be introduced to the game before playing against someone. This will ensure that their first game is less confusing and more fun. The tutorial is split in two parts to reflect survey results; many people are already familiar with Ultimate Noughts and Crosses and thus do not need a tutorial for that part, only for the new mechanics added by UTTTTT. This makes sure that the tutorial suits every players needs and does not feel redundant and unnecessary.	D

		1.4	**Sound options**	The options menu will include the ability to change volume settings.	This will allow the user to change the volume to fit their needs. As sound has been marked as an optional feature (see 4.2, 4.3), this is also an optional feature. However it would become essential if I am to implement sound so that if a user finds the sound distracting, they are able to turn it down or off so they have a better experience.	O
		1.5	**Credits**	Credits will include the name of everyone who helped with development of the original UTTTTT.	In each game I researched there was a credits page. Although this project is solely my work, the original UTTTTT was developed by both me and others. Therefore, a section of the game crediting them will be a nice touch especially considering I am making this adaptation for them to play,	D
		1.6	**Animated background**	Background with moving elements themed to the game.	Both *5D Chess with Multiverse Time Travel* and *Turmoil* had themed menus with at least some movement. I would like to design something similar with the main menu representing aspects of UTTTTT, such as moving nought and cross symbols (or other custom pieces) in various colours. This is important as in both these games, the main menu acts as an introduction to the game.	O

		1.7	**Exit game button**	A button that ends the game when pressed.	This is present in all the games I researched. It will allow the user to easily exit the game when they want to, which someone may not be able to do if they are less familiar with games.	E
2	**Gameplay**	**2.1**	**Ultimate Noughts and Crosses**	The base Ultimate Noughts and Crosses gameplay.	This is the premise of the game and is necessary for the rest of the features to be added. Without this feature there would be no game for the player to play.	E
		2.2	**Time travel**	When a player is sent to an already taken tile, they are given the opportunity to time travel. This is either by creating a new timeline or moving a piece somewhere or somewhen else on the board. The layout of timelines will be heavily inspired by *5D Chess With Multiverse Time Travel*.	Time travel is the core aspect of UTTTTT and was the first mechanic developed for it. Without time travel, I cannot consider this project an adaptation of it; my project would not be successful. This is reflected in the survey results for question 2.	E
		2.3	**Shop & items**	Players receive coins through events and elsewhere throughout the game. They can then use these coins to purchase items that put them at an advantage or put their opponent at a disadvantage. I plan to include items from the original UTTTTT rather than design my own.	The in-game shop and items are also core mechanics of UTTTTT. Including items from the original game will also make my adaptation more faithful to the actual game.	E

		2.4	**Events**	After a player's turn, an event may occur. This will either be a quick-time event where a player has to answer a question or play a minigame, or an event card where there may be some effect on the game. I plan to include events from the original UTTTTT rather than design my own.	Similarly to the shop, the events are an important part of UTTTTT and therefore essential for my adaptation. It also is required for the karma system to have an effect, which was deemed very important by question 2 in the survey.	E
		2.5	**Battleship**	Before the game starts, each player places their battleships on the board. A player's battleship is hit when their opponent places a piece on that square. A player loses if all their battleships are sunk.	This is a less important mechanic than the ones listed prior. It does not affect gameplay much and in the original UTTTTT, usually just acts as a way for players to randomly get extra money or experience points. It is also less desired as shown in the survey, being voted preferable to have but not needed. Therefore, although it is desirable, I do not consider it an essential feature.	D
		2.6	**Levelling system**	Players gain EXP over time. Players gain awards when they level up.	The levelling system is not a huge part of the original UTTTTT, mainly existing to give players extra coins and free items. However, according to survey results, it seems appealing to people unfamiliar to the game. Therefore, I will prioritise it somewhat as this aids in my goal of using my project to introduce more people to the game.	D

		2.7	**Karma**	Players gain positive or negative karma depending on their actions towards their opponent. This influences what kind of events they get. Players are not shown their karma value.	The karma system was considered a must-have in question 2, therefore I will prioritise including it. This is again to make my project a more accurate recreation of the original game.	E
		2.8	**Social deduction**	One of the player's pieces has a random chance of being the Demon. Each turn, a player may 'execute' one of their pieces in the present, removing it from the game. If a player executes their Demon piece, they win.	Social deduction was considered not needed in question 2. Additionally, the majority of people unfamiliar with the game did not find it very appealing. I think this would be an acceptable feature to omit for my adaptation. Furthermore, it is not a very well-developed mechanic in the original UTTTTT and is not used in some games.	O
		2.9	**Weather**	Weather conditions include sunny, rainy, cloudy and thunderous. Certain event cards may trigger depending on the weather and passive effects may be caused by certain weather conditions.	Similarly to social deduction, weather was considered both not needed and not very enjoyable. Therefore, the success of my project will not be reduced if it is not included.	O
		2.10	**Lifespan**	Each player starts with a random lifespan between 70-90. A player's lifespan decreases by 1 each turn or with events. When a player's lifespan runs out, they lose.	Lifespan faced mixed results in question 2, with one 'not needed' response and two 'must have' responses. As there is not a consensus on its importance, I do not think it is essential for the adaptation, so I will not prioritise adding it. Furthermore, survey results show it is not that appealing for newcomers to the game.	O

		2.11	**Custom pieces**	Before starting the game, players can design their own pieces to use instead of the typical nought and cross.	Like battleship, custom pieces were considered preferable to have in question 2. However, in question 4, it was overwhelmingly (77.8%) voted as very enjoyable. This means that adding it will make my project much more appealing to the people I want to introduce it to, making my project more successful. It is also a simple feature and will not take much time away from adding other features. Therefore, I will ensure that I include it.	E
		2.12	**Stocks**	As part of the in-game shop, a player may invest in stocks. They decide for how long they want to invest. A random number is generated then multiplied by the number of turns that the player wants to invest. This is the amount of coins the player gains or loses.	Stocks had mixed results in question 2, much like lifespan. Additionally, it was considered unenjoyable for those I want to introduce the game to. This means that adding it may discourage people from trying the game, which is the opposite of what I want. Therefore, I have marked it as an optional feature to include in my project.	O
		2.13	**Meow button**	A key that, when pressed, produces a meow sound effect.	The meow button was a feature requested in the survey. Although it is not a feature in the original UTTTT, I don't believe adding this button will change the game to a degree that makes my project a less successful adaptation. Although it is not essential, I think adding it would also make the game more appealing to new players as it is a fun extra feature.	D

			Hotkeys	Keys that perform actions when pressed.	Hotkeys will make performing actions easier as it avoids the need to move the mouse over to a button and click it. This will make gameplay quicker for experienced players and therefore make playing UTTTTT easier for them.	
4	**Miscellaneous**	**4.1**	**Multiplayer**	Multiplayer between two players. This can be done locally on one device or online between two devices.	The purpose of my project is to facilitate playing UTTTTT with others, therefore multiplayer is essential to achieve this purpose. I will ensure that I add at least one form of multiplayer to ensure this. Including both will be preferable so multiplayer is more accessible, but not necessary.	E
		4.2	**Sound effects**	Sound effects for various actions (clicking buttons, purchasing from the shop, placing pieces, etc.)	Sound effects are present in each of my researched games. I believe they would add engagement and immersion into the game and make playing more fun. However, they are not necessary for enjoyment, so I have marked these as optional.	O
		4.3	**Music**	Background music.	Music is present in most games as a way to add atmosphere and engagement. However, it is less necessary for my project as an adaptation of a pre-existing game, so the addition of music would be an optional feature.	O

		4.4	**Win streak**	Keep track of how many games a player has won in a row.	In the survey, a respondent expressed a wish for a win streak. Adding it would make the game feel more personal which is appealing. However, I do not find adding this feature high priority as only one respondent expressed a desire for this.	O

Limitations

Due to the scope of the project and time constraints, I have decided to put limitations on the project in multiple areas.

Multiplayer

To begin with, I will limit myself to implementing only one form of multiplayer. This is because although it is essential for this project to allow for multiplayer, implementing both is not strictly necessarily. I would like to instead focus on developing gameplay, as multiplayer is less important if there is not a game to play. I will only be implementing remote multiplayer, as on question 6 and 7 of the survey, results showed that remote multiplayer was slightly preferable. Although leaving out one form of multiplayer will make the game less accessible as remote multiplayer requires Internet and a computer each, I don't think this will cause too much issue. This is because all my main stakeholders have access to a computer and Internet, and likely so do most of the people I plan to introduce to UTTTTT with this project.

Single-player

A single-player mode is not necessary as my project is primarily intended to make multiplayer play of UTTTTT easier. While it could be an enjoyable feature and is present in other board game adaptations such as *5D Chess With Ultimate Time Travel*, Survey results indicate mixed feelings towards single-player. Additionally, it would take a very long time for me to develop; time which could be spent on features more suited to the goal of this project. Therefore, I will not be including singleplayer against an AI.

Number of gameplay mechanics

Ideally, all mechanics from the original UTTTTT would be implemented in my project. However, I will not have enough time to do this. Therefore, using the survey I have selected the features I believe are most important for a faithful recreation of UTTTTT. I will prioritise the development of these features first and then develop less important features second. This means that even in my limited time, I can still make a good adaptation.

Events

In the original UTTTTT, there are a large number of possible events that can occur. It will not be possible to implement all these events alongside all the other essential features I plan to include due to time constraints. This means I will prioritise adding simpler events such as the event cards or multiple-choice questions over more complex minigames. Although this will make events less accurate to the original game, it will allow me to have more time to work on other mechanics, which I believe is a fair trade-off.

Platforms

I will only be developing my project for one platform. As I have more experience developing desktop programs, I will begin by developing my project for desktop. This experience will mean I will be able to spend more time in development adding features to the game, rather than learning how to develop for an unfamiliar device. Additionally, my project has many features that will be better if viewed on a large screen. Mobile devices typically have small screens which makes them less suited for the game. I feel that one platform is enough because most people have access to a desktop.

Hardware & Software Requirements

Hardware

Any general-purpose computer with mouse input in order to interact with buttons and the game itself. Keyboard is needed for hotkeys but is not necessary for gameplay as all hotkey functions can be done by clicking buttons. A speaker or other audio output device is needed to hear music or sound effects but are not necessary to play as the audio only exists to add ambiance.

The computer will not need to be particularly powerful or need large storage or memory.

Software

There are no specific software requirements, and I will be able to port to both Windows and macOS.

Computational methods

This problem can be solved computationally in many ways. For example:

- The original UTTTTT suffers from needing a third player to manage the extra mechanics. My program aims to act as this third player through **automation**. This means that implementation of mechanics will be carried out by the program using algorithms, which removes human bias and error from the game. This will allow for a more enjoyable experience.

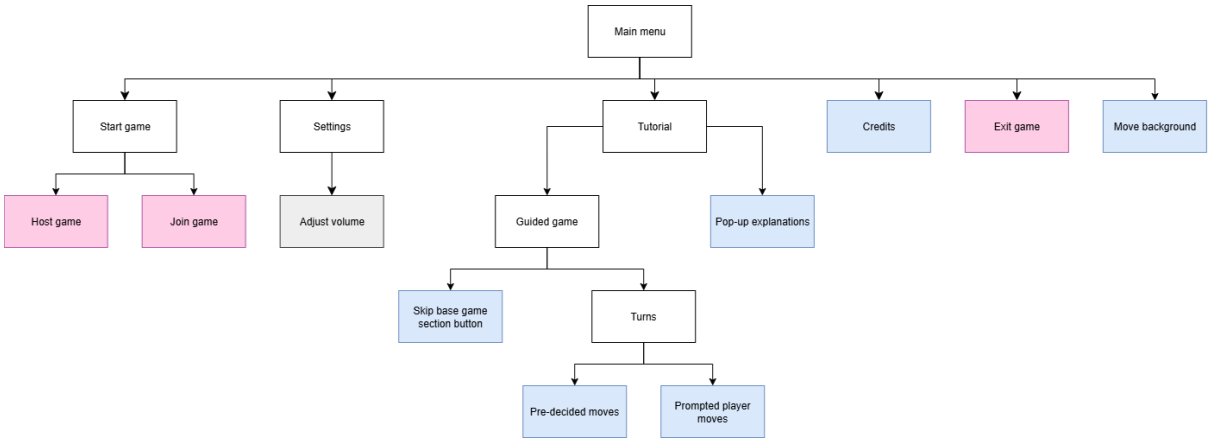
- UTTTTT contains many different mechanics that are separate from one another. This allows for easy **decomposition** as each mechanic can be tackled and broken down separately. This makes the solution manageable and easy to implement.
- The use of a GUI allows for **visualisation**. For example, the items in the shop can be displayed with icons. This will make it easier for players to see and understand what is going on compared to the base game that has no UI.

Design

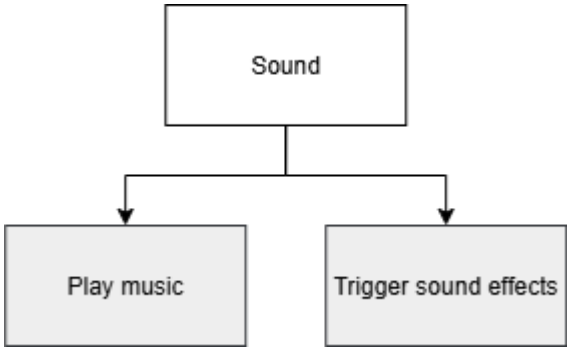
Structure Diagram

Stage 1: Pink
Stage 2: Red
Stage 3: Orange
Stage 4: Yellow
Stage 5: Green
Stage 6: Blue
Stage 7: Purple
Stage 8: Light grey
Stage 9: Dark grey

Main menu

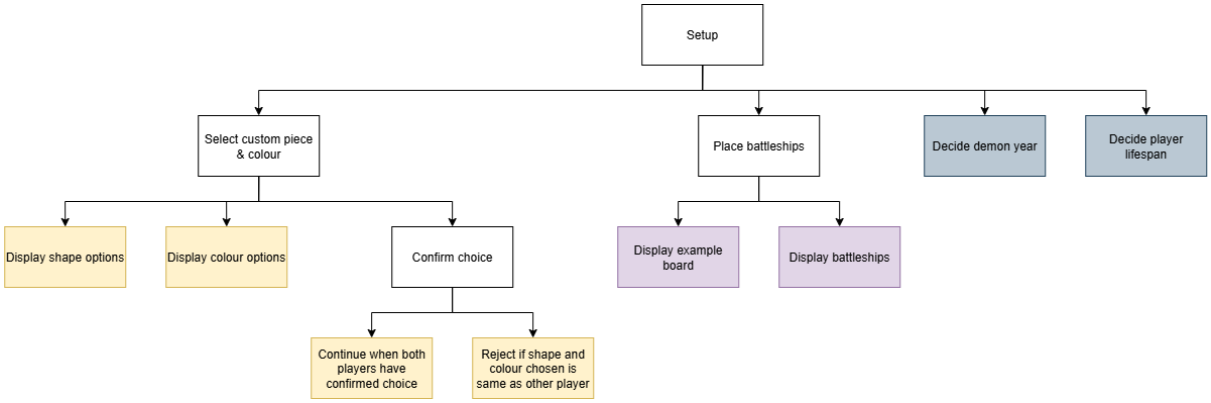


Sound



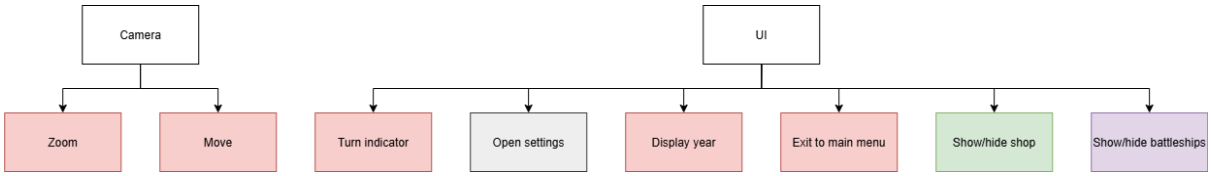
Gameplay

Setup



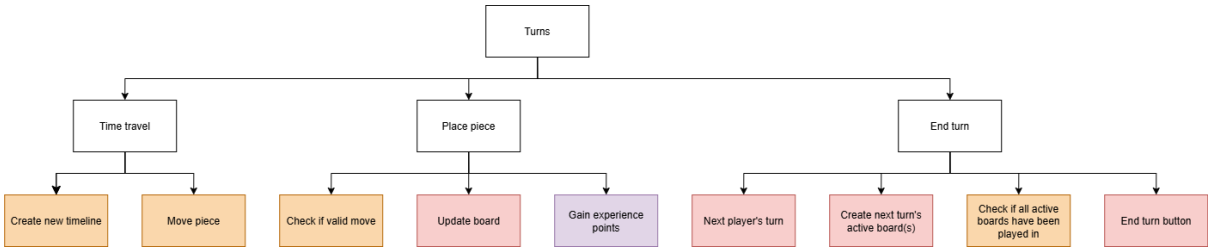
In-game

Camera & UI

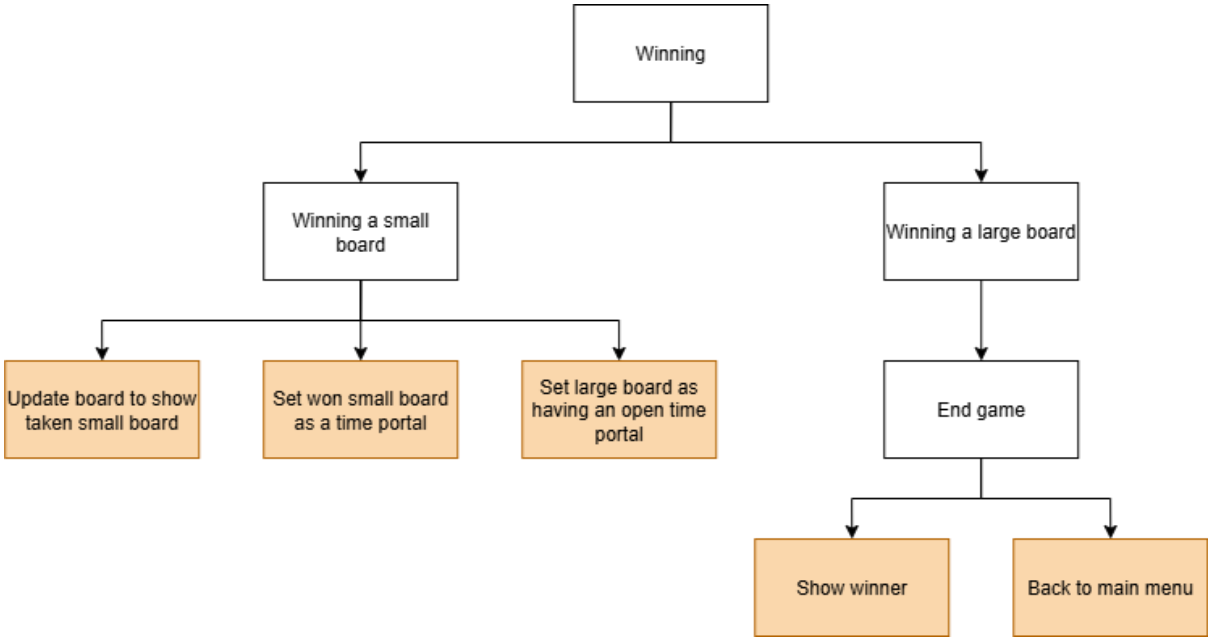


Base game

Turns

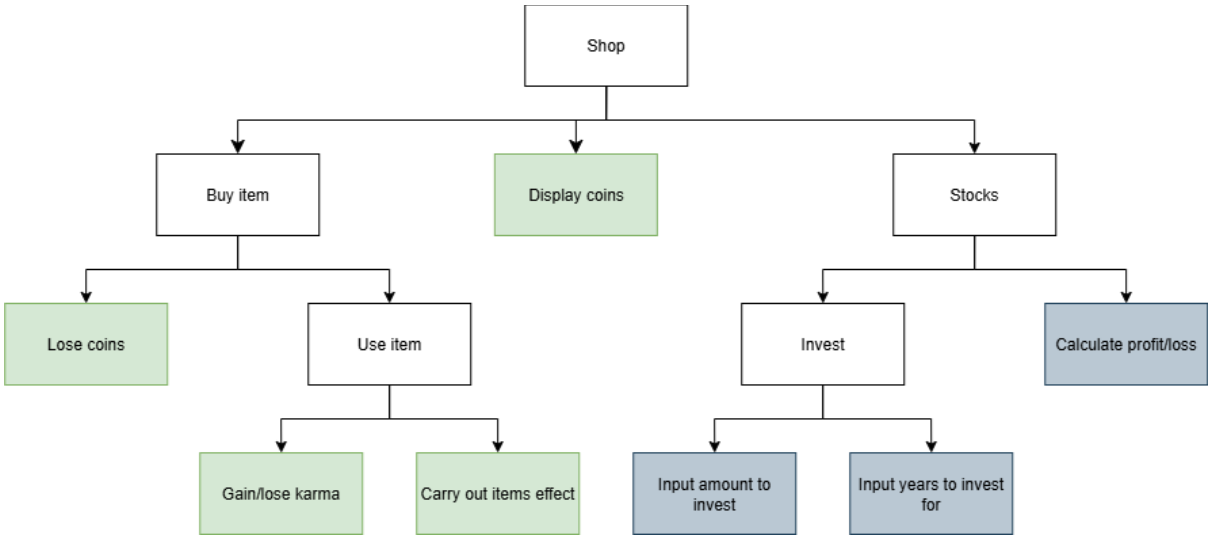


Winning

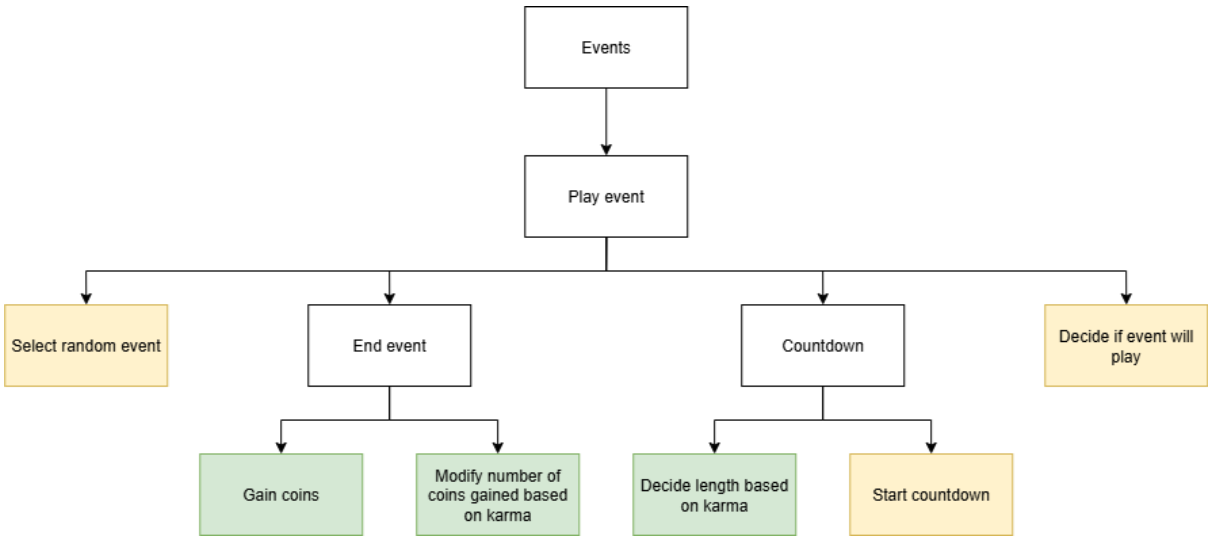


Added mechanics

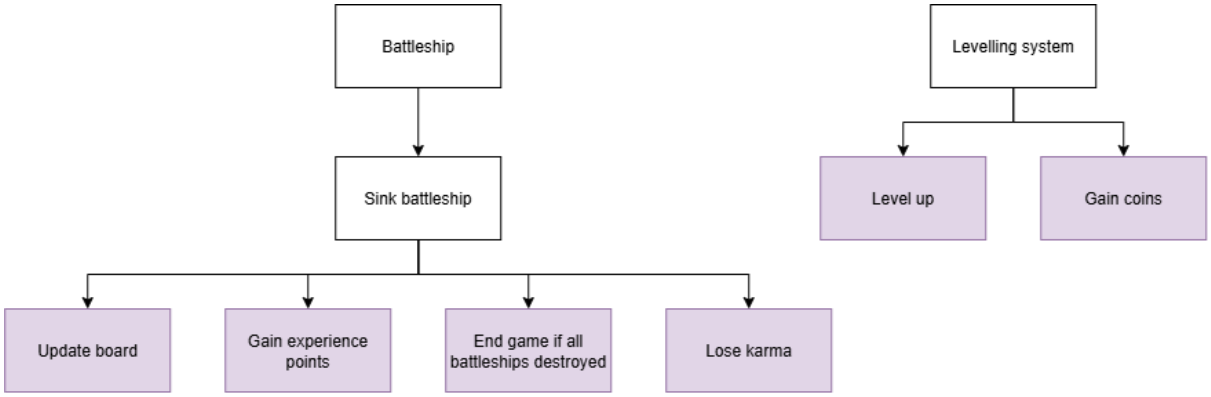
Shop



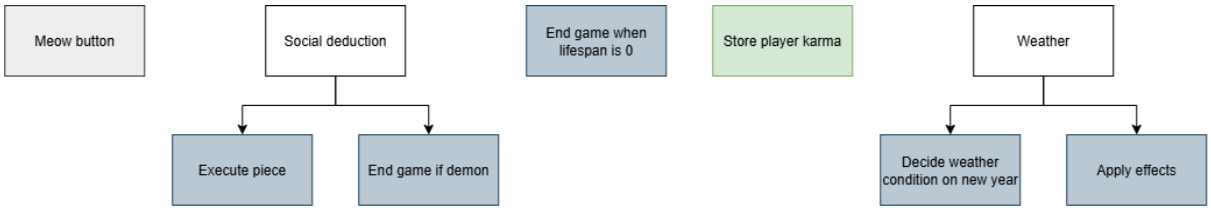
Events



Battleship & Levelling system



Miscellaneous



Stages of development

Stage 1 – Online multiplayer

Stage 1 involves implementing a basic main menu, from which the game can be exited or an online game can be hosted or joined. Two players will be able to connect to each other by having one host a game and the other join. When a player hosts a game, the game should start when a second player joins.

#	Description	Test data	Expected result	Actual result
1	Does pressing the exit button	Click exit button	Game should close	

	exit the game?			
2	Can a player host a game?	Click host button	A game is hosted that can be joined	
3	Can a player join a game hosted by another player?	Host a game on one instance of the game. On another instance, attempt to join the game hosted by the first instance.	The two players are connected.	
4	On the second player joining the game, does the game start?	Same as above.	The game transitions from the hosting/joining screen into the game	

Stage 2 – Turn-taking

Stage 2 creates the foundation of the game, involving the ability to move the camera around the game and take turns. This is done first so that the rest of the features can be added.

In stage 2, two players in a game should be able to take turns, move the camera around and exit the game. The game should be presented as one timeline. In a turn, a player should be able to place any number of pieces anywhere in the active board. A player should be able to press the end turn button to end their turn which will cause the next active board to be created. Additionally, this stage will include the in-game UI elements of the player whose turn it is.

#	Description	Test data	Expected result	Actual result
1a	Can the camera be moved?	Click and drag on screen	The camera moves with mouse movement	
1b	Does the camera zoom?	Scroll up/down	Camera zooms in/out respectively	
2a	Can a piece be placed?	Click on an empty small board square	A piece is placed there and is seen on both player's screens.	
2b	Does each player have a unique piece?	Click on an empty small board square	Player 1 (host) should place a cross, while player 2 (client) should place a nought.	

3a	Can a player end their turn after placing a piece?	Place a piece then press end turn.	A next active board should be created. The turn indicator should change	
3b	Can players only move on their turn?	Click on an empty small board square during the opposing player's turn	A piece cannot be placed	
3c	Can players only place a piece on the active board?	Click on an empty small board square on a non-active board	A piece cannot be placed	

Stage 3– Winning and time travel

Stage 3 is the rest of the base game, including time travel. Due to the existence of multiple timelines, time travel heavily affects the structure of the game in a way that it would be difficult to add in during later stages.

A player should be able to win by taking 3 small boards in a row. Additionally, move validation should be present. When a small board is taken, that point on the timeline should be saved as an open time portal. When a player is sent to a time portal at a later point, they should be able to either create a new timeline from a previously opened time portal or move one of their current pieces. A player should only be able to end their turn after they have played in each active board.

#	Description	Test data	Expected result	Actual result
1a	Can a small board be taken?	Place 3 pieces in a row as one player	A larger piece appears over that board	
1b	Can a taken small board not be played in?	Attempt to place a piece in a taken small board	No piece is placed	
1c	Does being sent to a taken small board trigger time travel?	Place a piece in a square in a small board that corresponds to the location of a taken small board	The next player is able to time travel	
2a	Are possible new timelines shown to a time	Send a player to a time portal	On that player's turn, they should be able to see a board coming off of each past board	

	travelling player?		with an open time portal.	
2b	Can a time travelling player create a new timeline?	Send a player to a time portal then click on a possible new timeline start point	A new active board is created and branch of timeline.	
3	Can a time travelling player move one of their pieces on the active board?	Send a player to a time portal then click on the piece to be moved. Then click on where to move it	The piece is moved	
4	Is a turn only able to be ended after a player places a piece on each active board?	Press end turn after not having placed a piece on each active board	The turn is not ended	

Stage 4 - Events

Stage 4 is the event system. After a player's turn there will be a small chance of a random event taking place, similar to a minigame. This stage takes place before the shop and karma system because the events are how players get coins to use in the shop, and the karma system only exists to affect how the event system acts.

When an event is triggered for a player, it is selected randomly from a list. A countdown from 10 will start and be visible. The event will only be visible and playable by the current player's turn.

#	Description	Test data	Expected result	Actual result
1a	Does the countdown appear?	Enter event	A countdown should appear in the top right	
1b	Does the countdown work?	Enter event	The countdown should reduce by 1 every second	
1c	Is an event ended after running out of time?	Enter event and wait for 10 seconds	At 0, the event is ended	
2a	Is an event ended after reaching the	Enter event, win or lose	The event is ended early	

	win or lose condition?			
3	Is the event only visible to the player in it?	Enter event	The other player's screen should not display the event	

Stage 5 - Shop and karma

Stage 5 is both the shop and karma system. The shop system is the first part of the stage and should be developed first as a player's karma is affected by the items they buy. The shop allows a player to use their coins to buy an item to use. Examples of items are gifts to the other player, an extra turn, a bomb that blows up the opponents pieces or a flamethrower that clears a small board. Karma is a hidden value that affects the number of coins received from an event. A player with low karma will get less reward.

#	Description	Test data	Expected result	Actual result
1a	Can the shop be opened?	Click the shop tab	The shop appears	
1b	Are coins displayed?	Click the shop tab	The player's number of coins are displayed	
1c	Are coins correctly stored?	Gain coins, click the shop tab	The number of coins should be correctly updated	
2	Can items be bought?	Click on an item in the shop tab	The correct number of coins should be taken away	
3	Can items be used?	Purchase an item then use it	The correct effect of the item should happen	
4	Does karma affect event rewards?	Win event with low/high karma	Coins gained should be lower/higher than usual	

Stage 6 - Tutorial

After all the essential features (stages 1-5) are added, development on requirement marked as desirable starts. Stage 6 includes expansion of the main menu with a tutorial and credits, the order of which doesn't matter. The tutorial will be a guided walkthrough with added rules explanation. It will be in two parts, with the first part being a walkthrough of a typical game of Ultimate Noughts and Crosses and the second part going through each added feature in the game with the exception of karma, which is a hidden mechanic. In the tutorial, users will not be able to make their own decisions and instead follow prompts. The credits will be a screen containing a list of people who

helped make the original UTTTTT. The main menu's animated background will also be added in this stage.

#	Description	Test data	Expected result	Actual result
1	Does the background move?	Start game	The background moves	
2a	Is the first part of the tutorial skippable?	Start tutorial, press skip button	The tutorial skips to the next section	
2b	Are player actions restricted in the tutorial?	Start tutorial, attempt to do something that is not prompted	Nothing happens	
3	Do credits open?	Press credits button	Credits open	

Stage 7 – Battleship and levels

Desirable game mechanics are added in stage 7. This stage includes the addition of battleship and the levelling system. Before the game starts, each player places battleships on the board. These are hidden when the game starts. If a player places a piece on top of an opponents battleship, that part of the battleship is destroyed. This gives the player coins. Destroying all of a player's battleships is an alternative way of winning the game. Levels are gained through experience points (EXP). EXP is gained through succeeding in events, destroying battleships and by finishing a turn. When a player levels up, they gain coins. The tutorial is also expanded to cover these features.

#	Description	Test data	Expected result	Actual result
1a	Is a player's level displayed correctly?	Start a game and gain a level	The display updates accordingly	
1b	Are coins given when levelling up?	Start a game and gain a level	Player's coin count increases	
2a	Before the start of the game, are players prompted to place battleships?	Start game	Battleship placing occurs	

2b	Can placed battleships be destroyed?	Place a piece over a player's placed battleship	That section of the battleship is destroyed	
2c	Can placed battleships only be destroyed by the opposing player?	Place a piece over your own placed battleship	Nothing happens	
2d	Does destroying battleships give EXP?	Destroy a battleship, check EXP	It increases	

Stage 8 – Sound

Stage 8 adds music, sound effects and the meow button. Music for the main menu and in-game will be added, as well as sound effects when clicking buttons, placing pieces, ending a turn and creating a time portal. This stage is optional as it only exists to add polish to the game and therefore is done only after desirable features are added. Stage 7 also adds settings which are used to control volume.

#	Description	Test data	Expected result	Actual result
1a	Do sound effects trigger?	Press a button	The button press sound effect should be heard	
1b	Are sound effects heard by the correct player?	Start a game, press a button	The sound effect is only heard by the player who pressed it	
2	Does music play?	Start the game	Music should be heard	
3	Can volume be adjusted using settings?	Start the game, change sound settings	Volume should change accordingly	

Stage 9 – Social deduction, lifespan, weather and stocks

Stage 9 adds the mechanics considered optional; social deduction, lifespan, weather and stocks, and their respective places in the tutorial. These features are not integral to any other stage or the success of my project and are not strongly desired according to stakeholders, therefore this is the final stage.

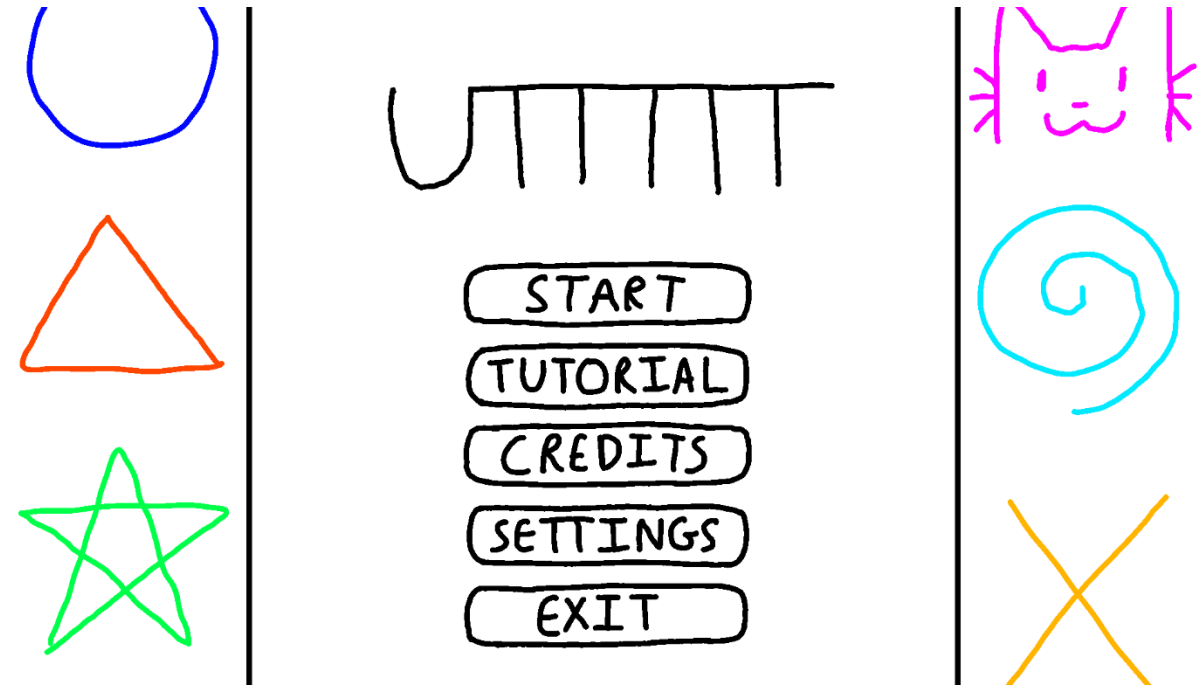
#	Description	Test data	Expected result	Actual result

1a	Can pieces be executed?	Start a game and press the execute button, then click a piece	It is deleted	
1b	Does executing the demon piece end the game?	Start a game and press the execute button, click the demon piece	It is deleted and the game ends with the player's win	
2	Does a player lose when their lifespan hits 0?	Start a game and play for the length of a player's lifespan	The game should end with the player's loss	
3	Are stocks invested for the correct amount of time?	Start a game and invest any amount in stocks for a number of years	The stocks should be returned after that number of years have passed	

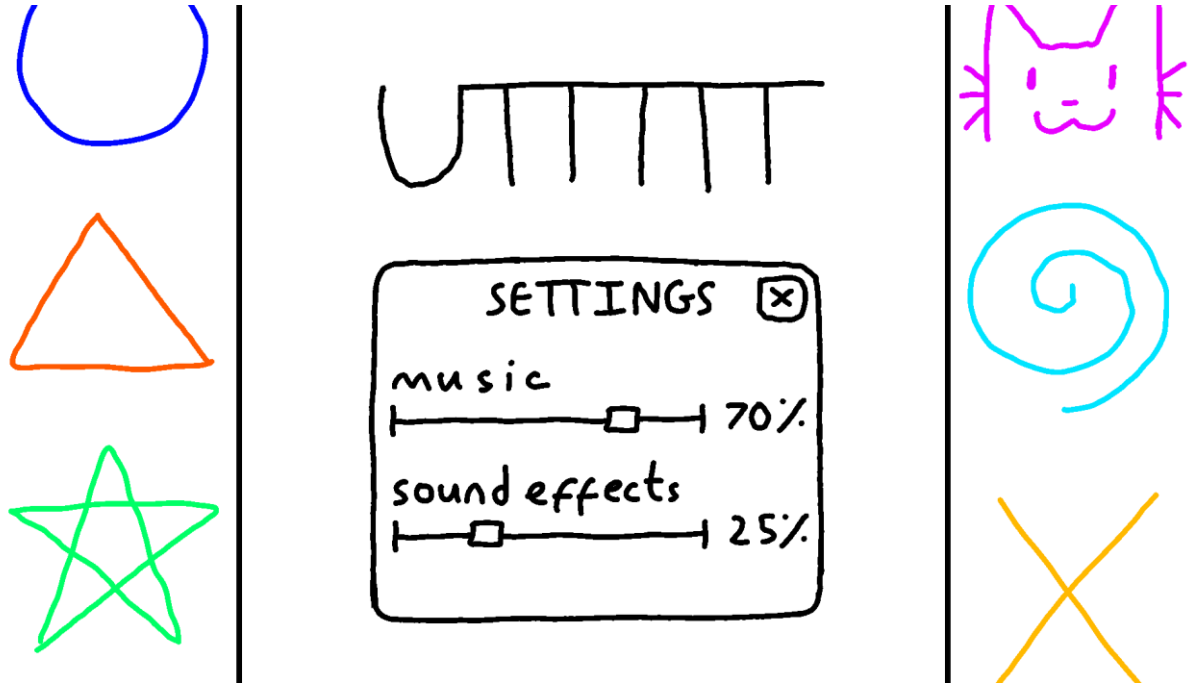
GUI Designs

All text and shapes in my game will be hand-drawn to give a sense of informality. This fits well with my project because it is an adaptation of a game made by me and a few close friends.

Main menu

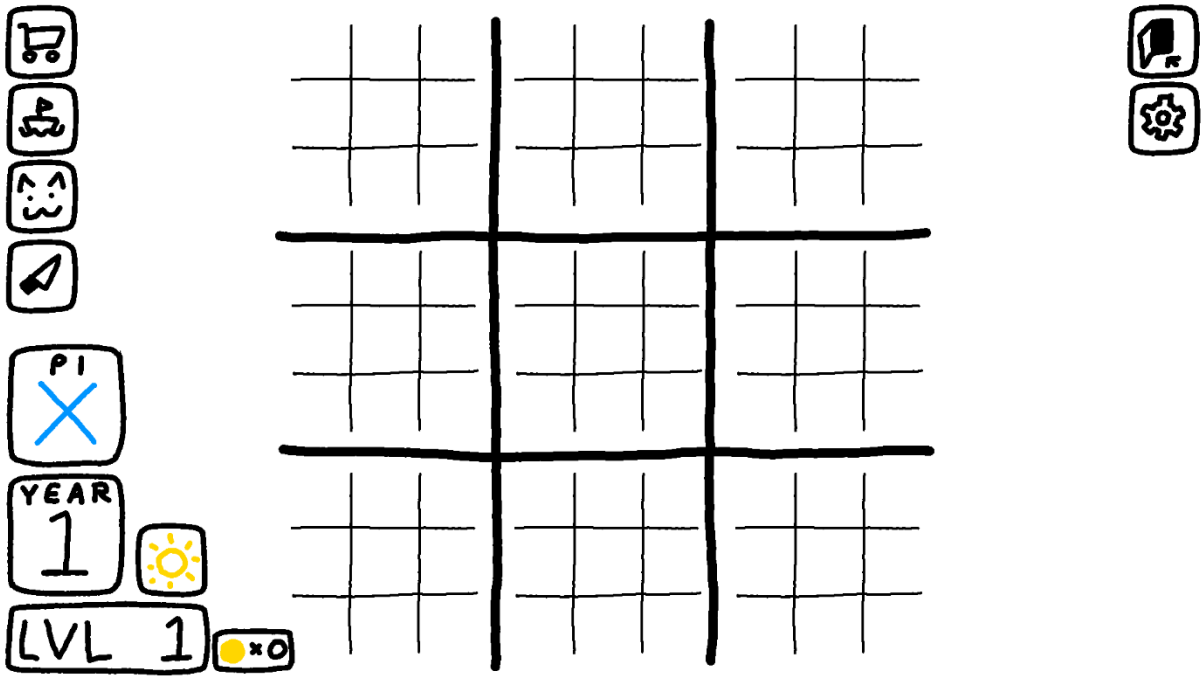


Symbols on the left and right will scroll down and up respectively and show different shapes and colours. This will add movement and playfulness to the main menu, which will keep players engaged.

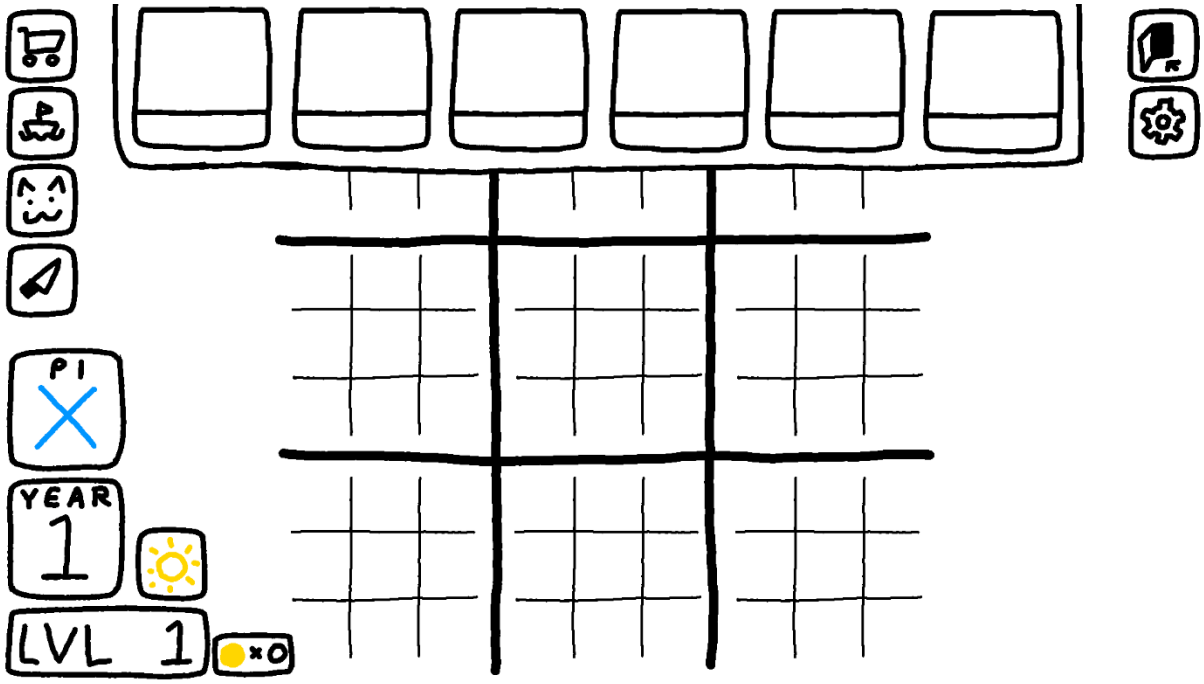


The settings will be a pop-up menu featuring sliders. This will allow users to change the volume of sound effects and music to the desired amount. The sliders give players a lot of freedom, improving player experience. Additionally, the percentages make it clear exactly how the volume has been modified. This is useful as it is not possible to tell exactly how volume has been changed through looking at the sliders alone.

In-game

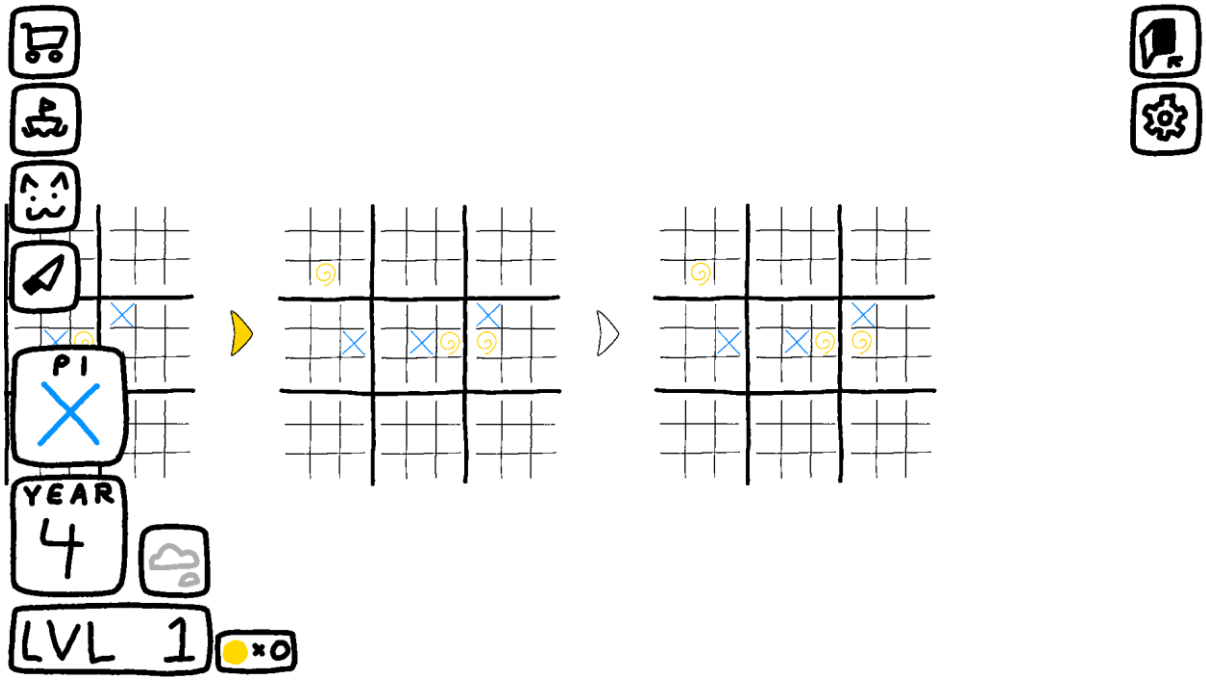


This is the UI of in an actual game as well as the tutorial. Each button (exit, settings, battleship, meow, execution and the shop) are clearly labelled with symbols, along with the weather and the current player whose turn it is. This clearly gives the player important information they need to know while saving space on the screen to see the boards and timeline. The player's level and the year's number is written large so it is easily visible. The board has varying line thickness, with the larger board having thicker lines and the smaller board having thinner lines, which makes it easier to see. This is



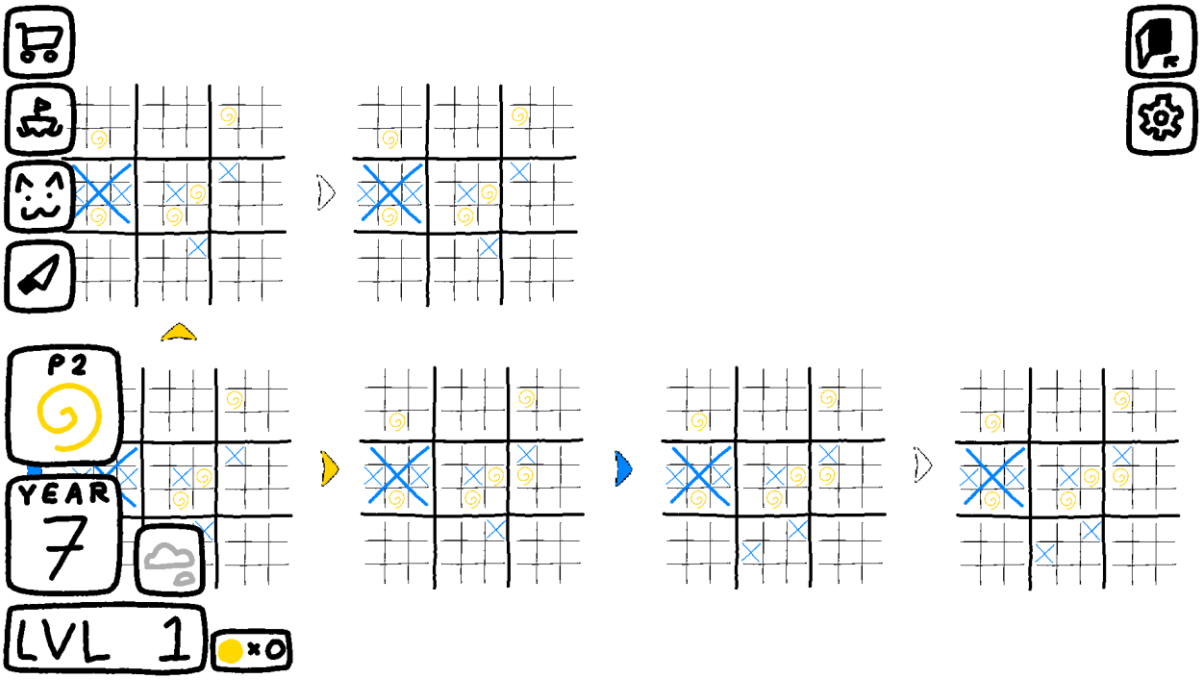
important in gameplay so that players can more easily tell small board apart from each other.

The shop opens at the top. This is so that a player can see what's in the shop while also seeing the current state of the game, which is important for deciding what item to purchase. In each box will be an item; the upper box will display an icon representing the item while the lower will show the price. This is so that it is quick to tell which item is which while also seeing how much they cost, something important for making decisions in-game. If stocks are added, they will be added here as one of the purchasable 'items'.



I have decided to place all UI pertinent to the game on the left because the timeline moves to the right. This means that players will often need to see the right side of the screen more than the left, so less on that side should be covered by UI. The exit game and settings button are on the right to make it clearer that they are apart from the other features and are not game mechanics.

Arrows point between boards showing the flow of time. Coloured arrows point to inactive boards in the past, and the colour corresponds to whose turn it was. Uncoloured arrows point to active boards to be played in. This makes it clear what boards need to be played in and what boards exist to look back in time, something which might be confusing without the visual indications.



Timelines are positioned in rows above or below one another. This, alongside the arrows, makes it easy for players to tell different timelines at a glance and which boards belong to which timeline, therefore making gameplay easier.

Development

Stage 1

Design

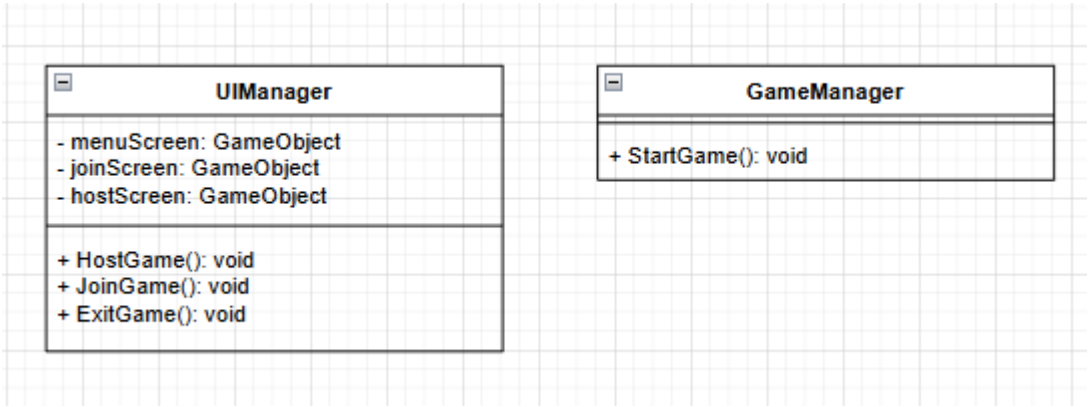
Stage 1 will include multiplayer functionality and the ability to end the game from the main menu.

Features to implement:

1. Display title, host button, join button and exit button
2. End game button ends game when pressed
3. Host button creates a game and join code
4. Host button changes the main menu to a screen displaying the join code

5. Join button changes the main menu to a screen where a join code can be inputted
6. Two users can connect to each other by having one host and the other join with the join code
7. On two players connecting, the game starts

Class diagram



To let me trigger code to run when a player clicks a button, I will make a UIManager object to which all methods intended to be run by pressing a button are stored. This is so all code to do with UI is stored in one place, making debugging easier.

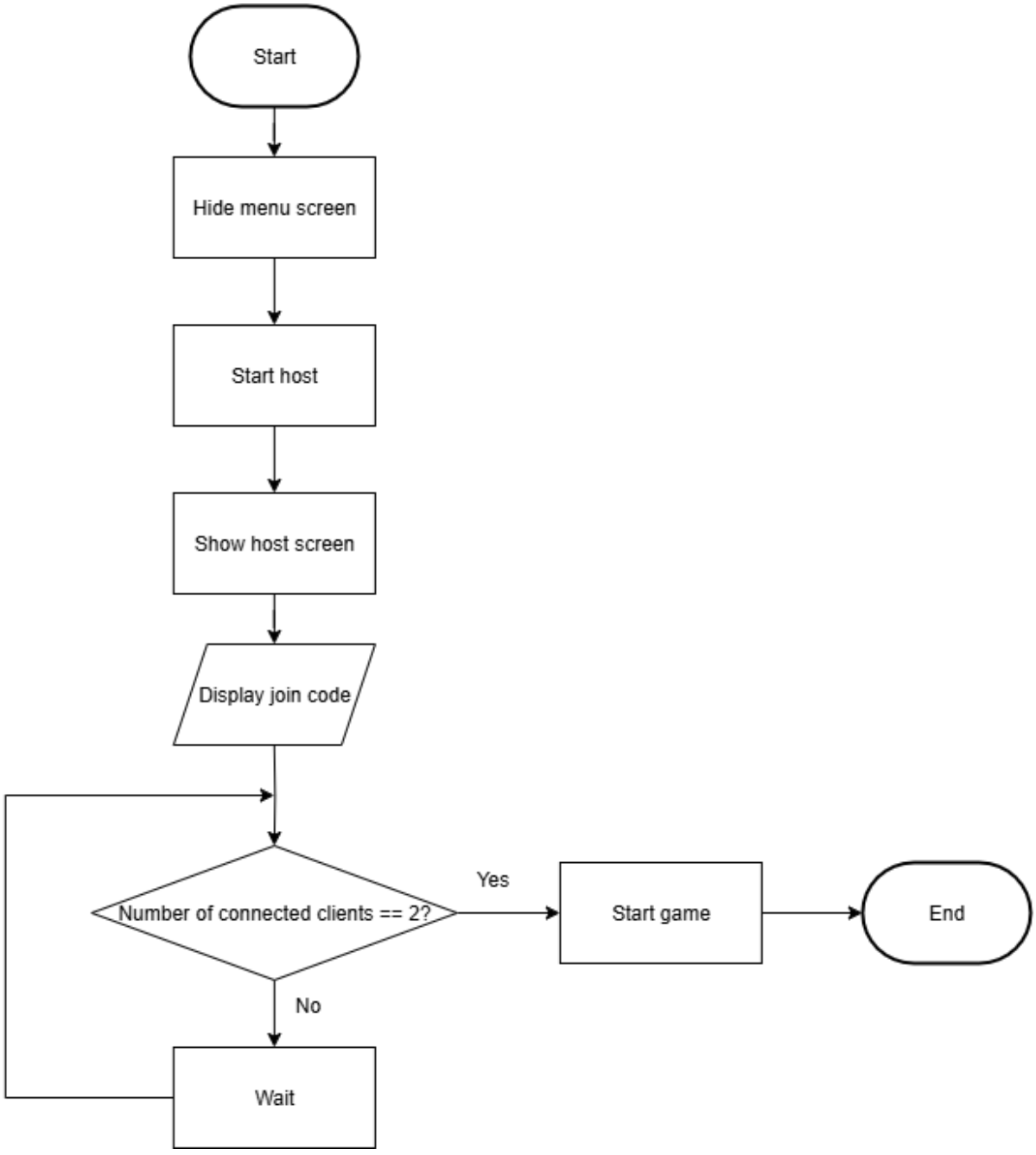
Data dictionary

Class	Variable	Data type	Description	Validation
UIManager	joinCode	string	Stores the join code used to join the game to be displayed on the host screen	Must be 6 characters long and characters must be alphanumeric. Alphabetical characters are uppercase.
	inputtedCode	string	Stores the code inputted by the player attempting to join	Must be 6 characters long and characters must be alphanumeric. Alphabetical characters are uppercase.
	menuScreen, hostScreen, joinScreen	GameObject	Each will be parents to different UI elements. On	N/A

			hiding one, all associated UI elements are hid. This allows for transitions between screens.	

Algorithms

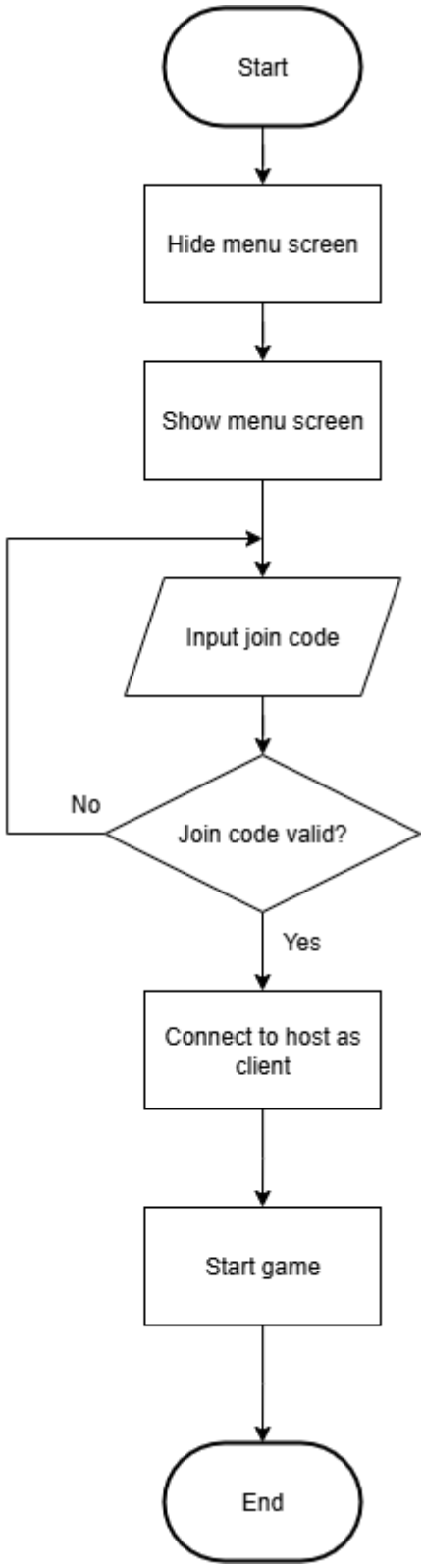
1. **HostGame() method** – will run upon host button being clicked. The algorithm will change the screen from the main menu to a 'host screen', where the game's join code will be displayed until the second player joins. When this happens, the game is started.



It is important that the screen is changed because it prevents the player from trying to host or join another game while they're already hosting one. This is important as it will not be possible to play in multiple games at once. I could also have achieved this by hiding the host and join buttons after it is pressed, however I believe it additionally makes more sense to switch screens entirely as it gives extra space to display the join code. This is important as the player needs to know the join code in order to share it with the person they're playing with.

2. **JoinGame() method -** will run upon the join button being clicked. This will change the screen from the main menu to a 'join screen', where a join code can be inputted.

When a valid join code is inputted, the player connects to the host and the game is started.



The screen is changed for the same reasons as in the previous method, in that it prevents a player from joining or hosting a game while attempting to join another.

3. **ExitGame()** – will run upon the exit button being clicked. The game will be quit using Unity's built-in Application.Quit() function.

```
PUBLIC PROCEDURE ExitGame()
      Application.Quit() //ends the game
ENDPROCEDURE
```

Development

To start with, I began with the main menu. I made a Main Menu scene and imported the sprites for the host button, join button, exit button and title and added them to the scene. The host button, join button and exit button are added as buttons using Unity's in-built UI elements while the title is just added as an image. I also install the Netcode for GameObjects package in order to later implement multiplayer.

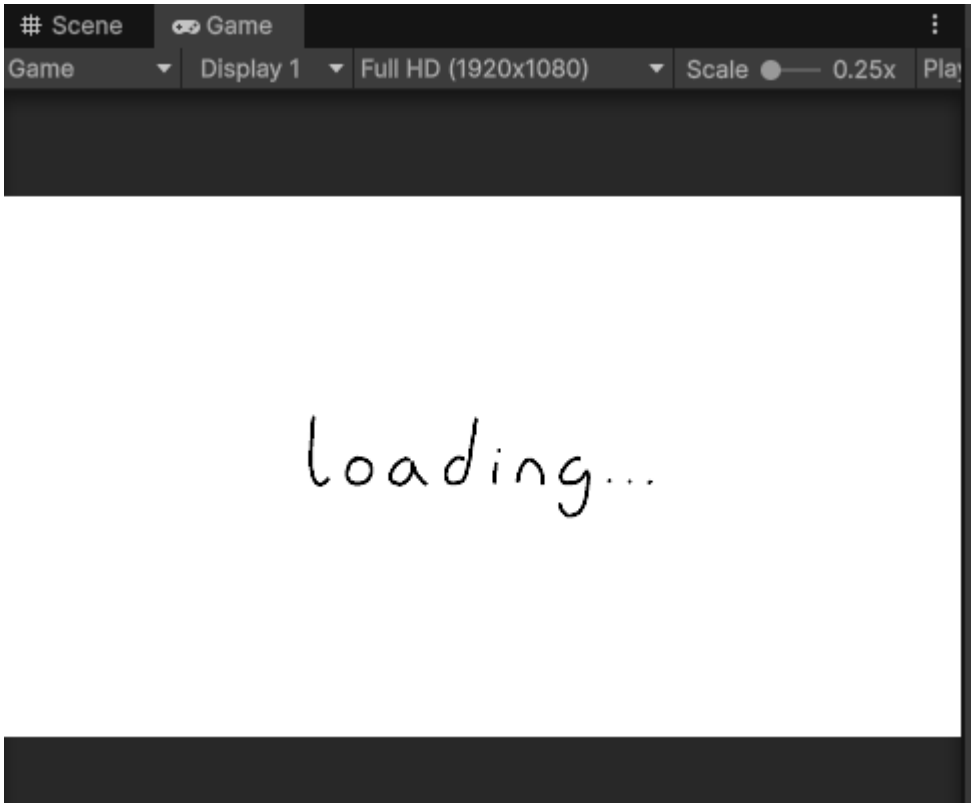
First, I added functionality to the exit game button. Below is the code to be run when the exit button is pressed.

```
public class UIManager : MonoBehaviour
{
    0 references
    public void ExitGame()
    {
        Application.Quit();
    }
```

I ran a build of the game, which was successfully closed when I clicked the Exit button.

Then, I started adding the functionality for two players to connect. I did this through using Unity Relay and Netcode for GameObjects. Relay will allow me to have two players to connect to each other using a join code while Netcode for GameObjects allows the logic of the game to work between the two players. Unity Lobby was another option I could have used alongside these two, which would have allowed users to create public lobbies that other players could search for and join. However, I decided against it because I intend for my game to be played between two people who already know each other. Therefore, the ability to search for games and join with random people is not something I need. In order to learn how to implement Unity Relay, I followed this tutorial from the Unity Documentation.

Before being able to use Unity Relay, authentication is required. I made a new scene called Init (for initialise) for this process because I wanted it to happen as soon as the game is started, before the player reaches the main menu. This is because my game is only playable online, therefore if authentication fails there is no point in letting the player play the game. I drew loading text to be displayed in this scene so it is clear that the game is loading. Below is the loading screen and the code for authenticating the user.




```
async void Start()
{
    await UnityServices.InitializeAsync();

    if (UnityServices.State == ServicesInitializationState.Initialized)
    {
        await AuthenticationService.Instance.SignInAnonymouslyAsync();

        SceneManager.LoadSceneAsync("Main Menu");
    }
}
```

When the game starts, this procedure is called. When the player is successfully authenticated, the scene should change to the main menu. This works as intended as when I ran the game, after a moment it changed to the main menu.

With authentication set up, I am able to start on using Relay to add functionality to the host and join button. I will start with the host button because I have to be able to host a game before joining it. I made a RelayManager GameObject and attached script in order to manage using Relay to host and join games. When the host button is clicked, it calls the StartHostWithRelay() method. Before adding any content to the method, I test whether the host button worked by programming it to output “host” in the console when clicked.

```
public void HostGame()
{
    Debug.Log("host");
}
```

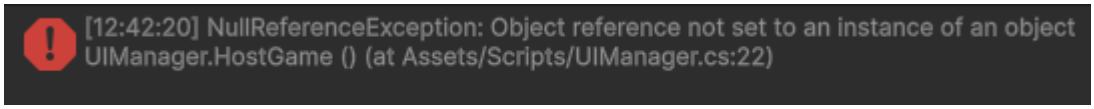
When run, the console successfully outputted the message.

Following the tutorial from earlier, I then added the code to create a relay and output the join code and updated HostGame() to call this new method.

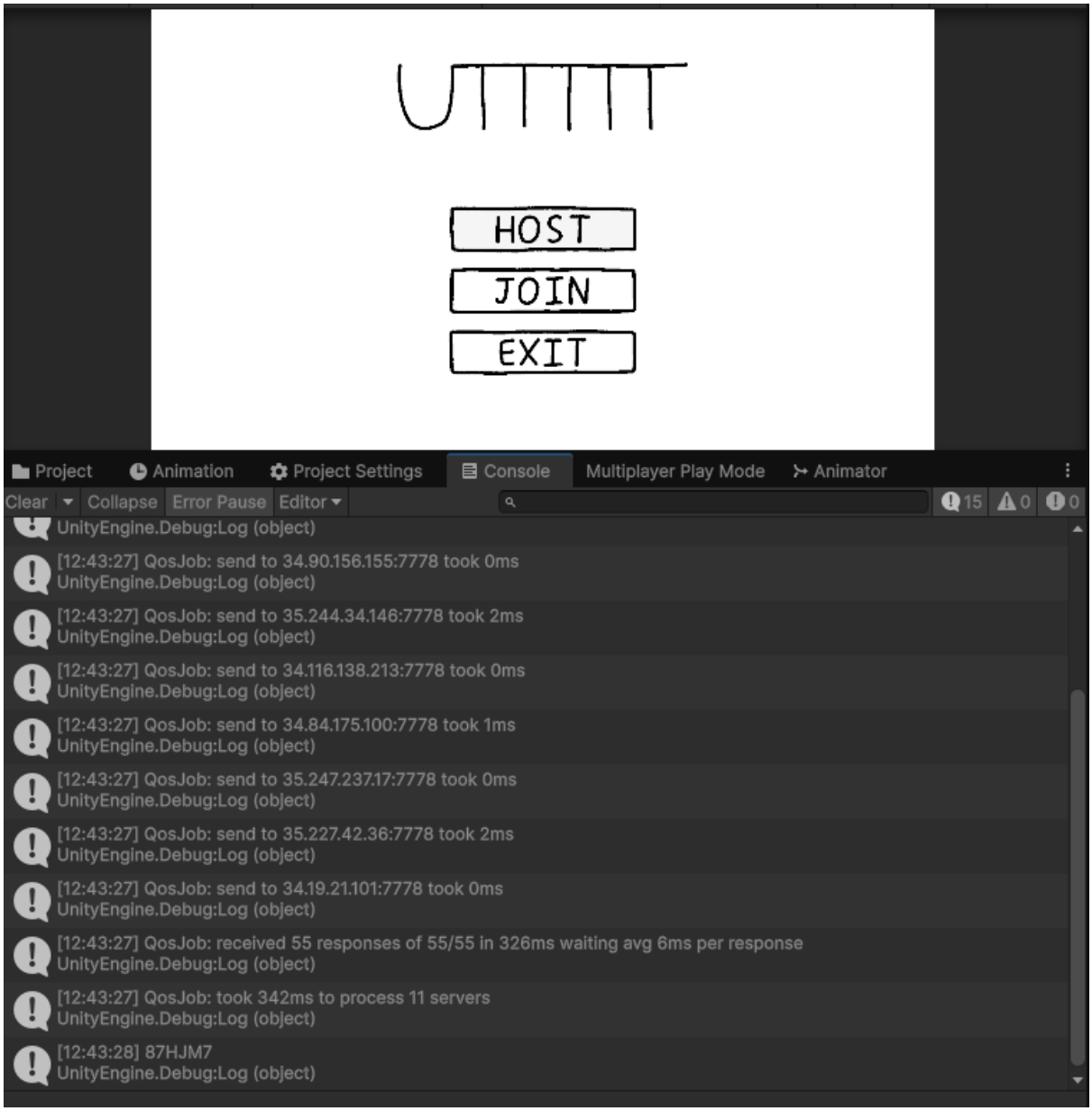
```
public async Task<string> StartHostWithRelay()
{
    //create an allocation for only one other peer; default region
    Allocation allocation = await RelayService.Instance.CreateAllocationAsync(1);
    //get allocation join code
    string joinCode = await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);

    //pass allocation to UnityTransport
    RelayServerData relayServerData = AllocationUtils.ToRelayServerData(allocation, "dtls");
    NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

    NetworkManager.Singleton.StartHost(); //start host

    return joinCode;
}
```

```
public async void HostGame()
{
    Debug.Log(await RelayManager.Instance.StartHostWithRelay());
}
```

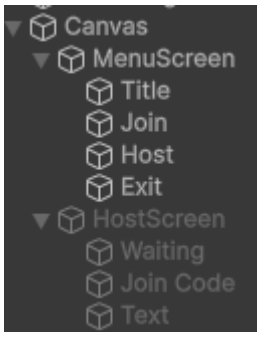
When I ran the project and attempted to click the host button, Unity displayed a NullReferenceException error.



This was because I had forgotten to attach the Relay manager script to the RelayManager game object. When I had done this and ran the project again, the join code was correctly outputted to the console.



Next I added the code to transition the host to a waiting screen where the join code would be displayed until the second player joins. I created the HostScreen and its UI elements.



Join Code has a TextMeshPro component. This will allow me to edit the text via a script, which will allow me to display the code. Although the font does not match the rest of the

handwritten text in the game, I think it is suitable as it ensures it is easy to read. This is important as the code must be readable to be shared to and entered by the other player. Below is the updated HostGame() code to include changing screens and displaying the join code.

```
public async void HostGame()
{
    menuScreen.SetActive(false);

    //sets code text to join code
    string joinCode = await RelayManager.Instance.StartHostWithRelay();
    codeDisplay.text = joinCode;

    hostScreen.SetActive(true);
}
```

When run and the host button is pressed, the screen changes and the join code is correctly displayed.

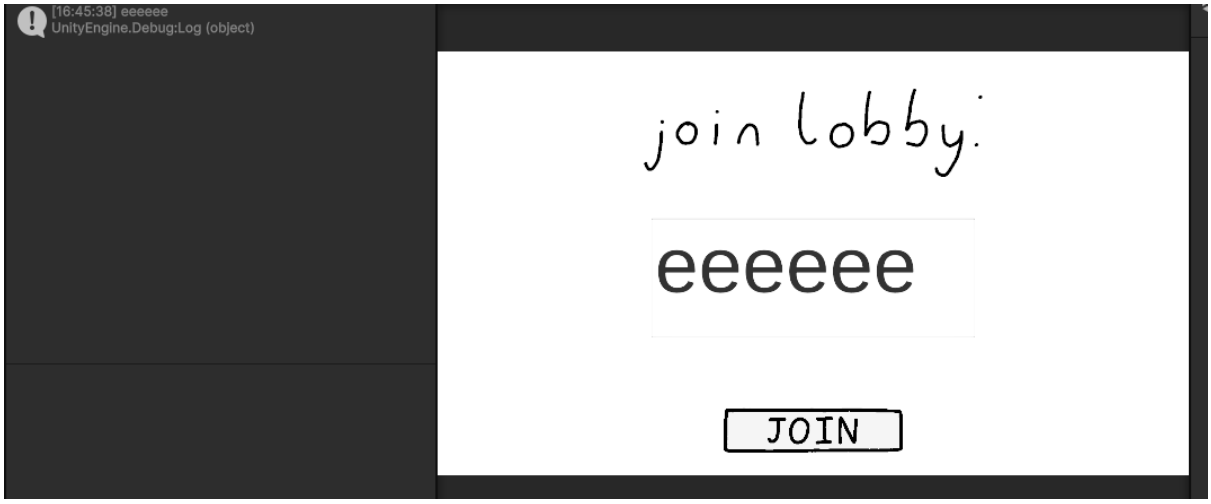
Now that a relay is successfully created, I start implementing the ability to join. I make a new join screen where a player can enter the join code using a text field and submit it by pressing a button.

```
public void JoinGame()
{
    menuScreen.SetActive(false);
    joinScreen.SetActive(true);
}
```

Pressing the join button changes the screen to the intended screen. I can also input text into the field, with a character limit of 6.

Next I write the code for the button to read the text in the input field. Included in this method is code to remove an extra character from the inputted text. This is because inputting text into a field in Unity adds an extra character, which will cause the join code to not be read properly by Relay if not removed. This will cause a player to be unable to join a game, so it is important this extra character is removed.

```
public async void EnterCode()
{
    string inputtedCode = codeInput.text;
    inputtedCode = inputtedCode.Substring(0, 6); //remove extra character
    Debug.Log(inputtedCode);
}
```



When I enter a code and press join, the entered text is successfully read.

However, with strings less than 6 characters long, an error occurs. To validate against this I make sure that the code length is checked before the extra character is removed.

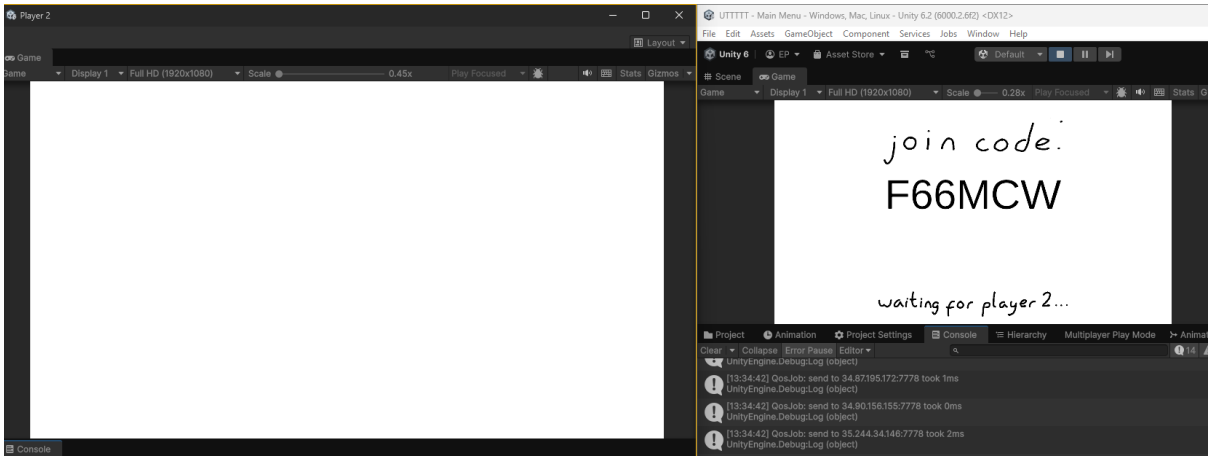
```
public async void EnterCode()
{
    string inputtedCode = codeInput.text;
    if (inputtedCode.Length != 7) return; //return if code is wrong length
    inputtedCode = inputtedCode.Substring(0, 6); //remove extra character

    if (await RelayManager.Instance.StartClientWithRelay(inputtedCode))
    {
        joinScreen.SetActive(false);
    }
}
```

Next I add joining the relay and starting the client. I update the EnterCode() method so that it calls the new StartClientWithRelay() method while passing through the inputted join code. If the relay is joined successfully, the join screen will be hidden.

```
public async void EnterCode()
{
    string inputtedCode = codeInput.text;
    inputtedCode = inputtedCode.Substring(0, 6); //remove extra character

    if (await RelayManager.Instance.StartClientWithRelay(inputtedCode))
    {
        joinScreen.SetActive(false);
    }
}
```

```
public async Task<bool> StartClientWithRelay(string joinCode)
{
    //join allocation with join code
    JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(joinCode);

    //set up transport
    RelayServerData relayServerData = AllocationUtils.ToRelayServerData(joinAllocation, "dtls");
    NetworkManager.Singleton.GetComponent<UnityTransport>().SetRelayServerData(relayServerData);

    NetworkManager.Singleton.StartClient(); //start client

    return true;
}
```

I ran the game simulating two players. On player 1, I hosted a game while on player 2 I joined the game I had hosted. On entering the join code, player 2's join screen disappeared, showing it had successfully connected to the relay.



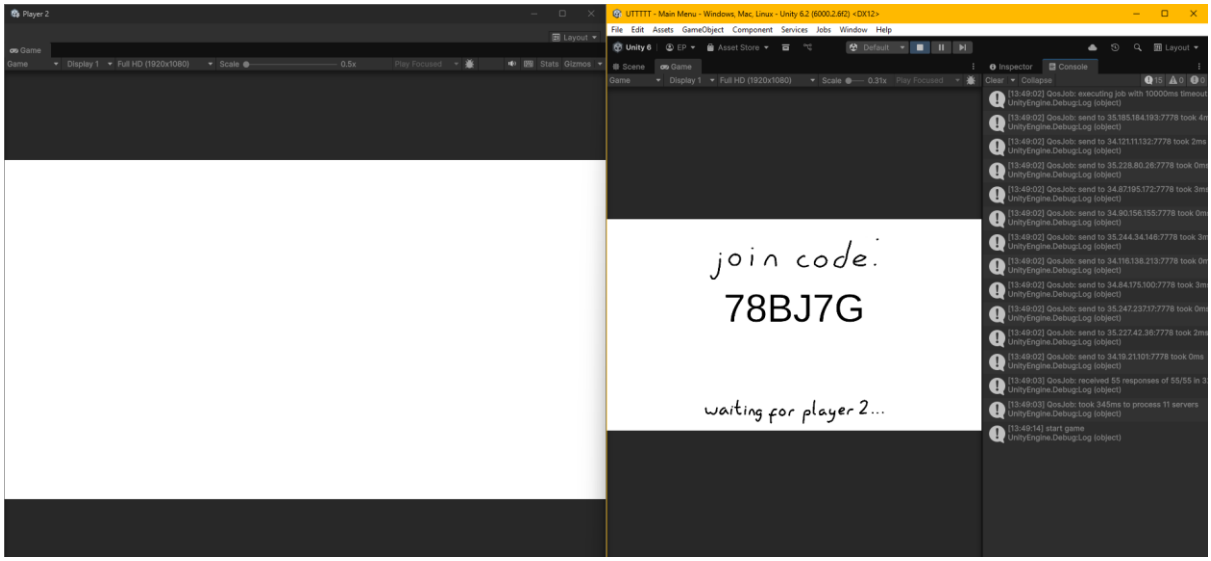
The last step of this stage is to transition to the game when two players have connected. To do this I make the GameManager. In order to know when to start the game, the GameManager subscribes to the OnClientConnectedCallback event. This will cause the OnClientConnected() method below to run, where I intend to add the code to start the game. To test that OnClientConnected() runs as intended, I just code it to output “start game”.

```
public override void OnNetworkSpawn()
{
    NetworkManager.Singleton.OnClientConnectedCallback += OnClientConnected;
}

0 references
public override void OnNetworkDespawn()
{
    NetworkManager.Singleton.OnClientConnectedCallback -= OnClientConnected;
}

2 references
private void OnClientConnected(ulong obj)
{
    Debug.Log("start game");
}
```

Start game was outputted before the second player had joined, instead when the first player had hosted the game which is not what I had intended. I realised that this had happened because when the host acts both as a server and a client, therefore OnClientConnected() is called when they join.

```
private void OnClientConnected(ulong obj)
{
    //start game if both players are connected
    if (NetworkManager.Singleton.ConnectedClientsList.Count == 2) Debug.Log("start game");
}
```

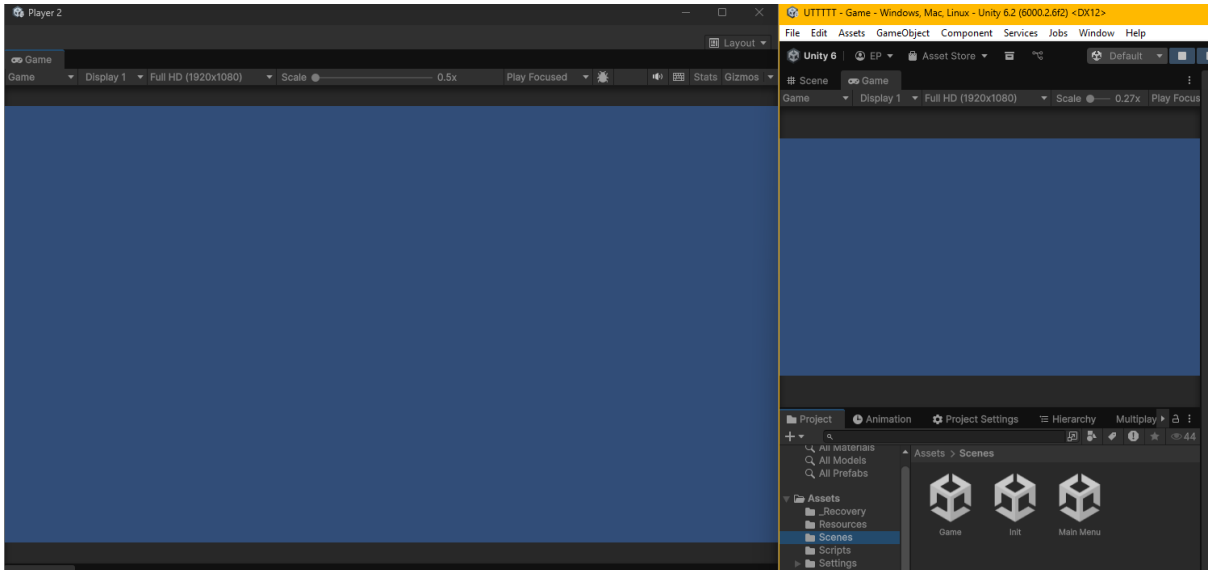
I fixed this by adding an if statement that ensures the code only runs when there are 2 clients in the game. On running this new code, “start game” is only outputted upon the second player joining.



Now that the time to start the game is being correctly detected, I will add the functionality to transition to the game. To do this I make a new empty scene called Game where the game will take place.

```
private void OnClientConnected(ulong obj)
{
    //start game if both players are connected
    if (NetworkManager.Singleton.ConnectedClientsList.Count == 2) SceneManager.LoadScene("Game");
}
```

However, when I hosted and joined a game, Unity outputted an error upon running this because I had forgotten to add the Game scene to the build profile. I did this, ran it again and it worked.



Testing & Review

The goal of this stage was to implement multiplayer and this goal has been reached. When a stakeholder played the game, they commented on the lack of back buttons on the host and join screens. Furthermore, joining and creating relays takes time, causing waits between changing screens. The screen is blank during these waits, meaning it is unclear to the player what is happening. This may cause impatient players to believe the game is broken and close it. I can remedy this by changing to a loading screen during these periods. To conclude, I believe this stage has been successful in achieving its goal and I am ready to move on to the next stage, however there are some features that can be added in order to improve user experience.

#	Description	Test data	Expected result	Actual result
1	Does pressing the exit button exit the game?	Click exit button	Game should close	PASSED Video 1

2	Can a player host a game?	Click host button	A game is hosted that can be joined.	PASSED *
3	Can a player join a game hosted by another player?	Host a game on one instance of the game. On another instance, attempt to join the game hosted by the first instance.	The two players are connected.	PASSED *
4	On the second player joining the game, does the game start?	Same as above.	The game transitions from the hosting/joining screen into the game	PASSED *

*Video evidence currently missing because I do not have a second PC to test the multiplayer on two builds of the game myself, so I have to ask a friend. I forgot to record the test the first time and my friend has been unavailable since. Video evidence will be added when I find another way to test it.

Stage 2

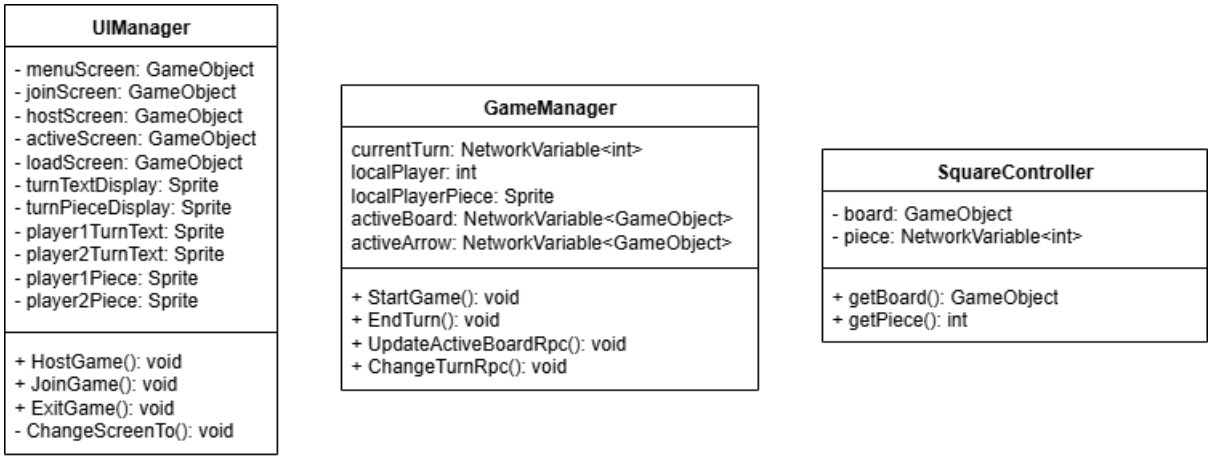
Stage 2 will improve upon stage 1 with loading screens and back buttons and will also add features to the game itself with camera movement, placing pieces, a timeline, turn-taking and UI.

Features to implement:

1. Loading screens as necessary
2. Back buttons on host and join game screens
3. In-game camera movement
4. Placing pieces with correct player display
5. Simple timeline
6. Turn-taking, including an end turn button
7. Turn indicator UI

Design

Class diagram



Data dictionary

Attribute of	Name	Data type	Description	Validation
GameManager	currentTurn	NetworkVariable<int>	Stores the number of the player whose turn it is.	Either 1 or 2.
	localPlayer	int	Stores the player number of the player on the local machine.	Either 1 or 2.
	localPlayerPiece	Sprite	Stores the sprite of the player of the local machine.	N/A
	activeBoard	NetworkVariable<GameObject>	Stores the current active board to be played in.	N/A
	activeArrow	NetworkVariable<GameObject>	Stores the current active arrow.	N/A

UIManager	activeScreen	GameObject	Stores the current screen being displayed.	N/A
	loadScreen	GameObject	Stores the empty parent of the UI elements of the loading screen.	N/A
	player1TurnText, player2TurnText	Sprite	Stores the sprites used in the turn indicator that state which player's turn it is.	N/A
	player1Piece, player2Piece	Sprite	Stores the sprite of the pieces for player 1 and player 2 respectively.	N/A
	turnTextDisplay, turnPieceDisplay	Sprite	Stores the sprite currently being displayed for the turn text and sprite respectively.	N/A
SquareController	board	GameObject	Stores the board that the square belongs to.	N/A
	piece	NetworkVariable<int>	Stores the piece that the square is storing.	N/A

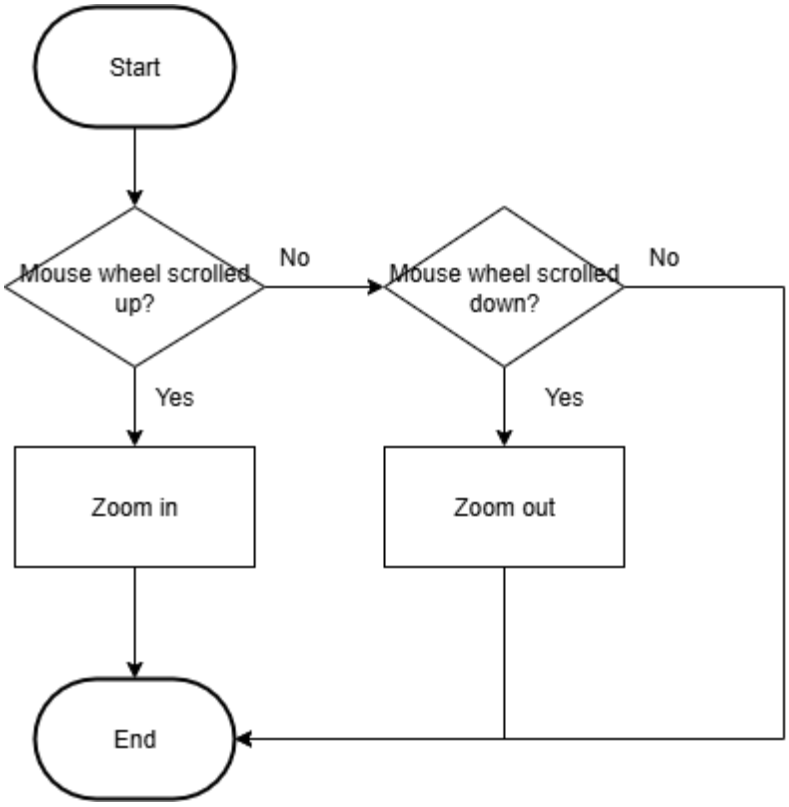
Algorithms

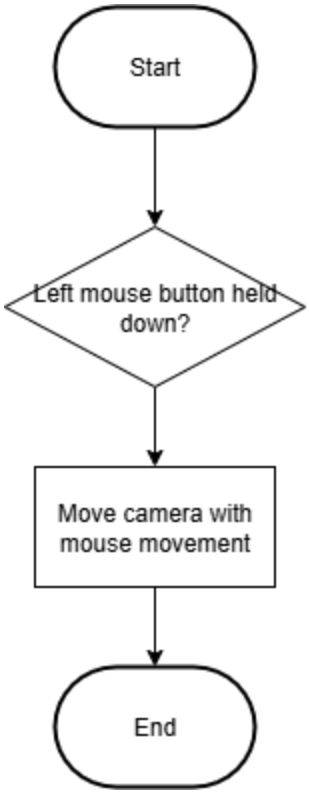
1. **ChangeScreenTo() method –** will be used in methods called when pressing buttons on the menu. It hides the current active screen and shows the screen passed in through the parameter.

```
PROCEDURE ChangeScreenTo(newScreen):
      activeScreen.hide() //hides current screen
      newScreen.show() //displays specified screen
ENDPROCEDURE
```

Implementing this method will make pre-existing code more efficient as I change between screens in the main menu often. Keeping this code in a method will reduce the need to repeat code and will also reduce the risk of errors as it ensures no mistakes can be made when rewriting code. This is especially important as I plan to add more loading screens and back buttons, so I will be using this method to do so.

2. **Camera movement method –** uses Unity's default method Update() that is called each frame. It will update the zoom of the camera with how much the scroll wheel has changed position and additionally update the position of the camera with mouse movement if the left mouse button is held down.





The first algorithm will control the zoom of the camera. When the player scrolls up, the camera will zoom in while when they scroll down, the camera will zoom out. The second algorithm controls camera movement. When the left mouse button is held down, the camera is moved alongwith cursor movement. This creates a dragging effect. Using the Update() function will allow for fast and responsive movement as it is called each frame. This means that movement will be smooth and not jarring, which is important as the player may need to move the camera around a lot as the game goes on. This will lead to a better playing experience.

3. **Placing pieces –** Due to the nature of network authority, this will require two methods, OnSquareClicked() and PlacePieceRpc().
   a. **OnSquareClicked()** – this will be called when a square is clicked. It will check that placing a piece there is valid.

```
PROCEDURE OnSquareClicked(clickedSquare)
      //checks that it is the player's turn, the square is empty,
and the board is active
      clickedBoard = clickedSquare.getBoard()
      IF currentTurn == localPlayer AND clickedSquare.getPiece == 0
AND clickedBoard == activeBoard.value
            clickedSquare.PlacePieceRpc(localPlayer,
localPlayerPieceSprite) //place the player's piece on the square
      ENDIF
ENDPROCEDURE
```

If the number of the player who clicked matches the number of the player whose turn it is, the square is empty and the square's board is the active board, then a piece will be

placed. This will make sure that a piece is only placed when it is supposed to be, which is when all three of these conditions are met.

b. **PlacePieceRpc()** – this will be called when a piece should be placed.

```
[Rpc(SendTo.Server)]
PROCEDURE PlacePieceRpc(player, sprite)
      piece.value = player
      display = sprite
ENDPROCEDURE
```

This method will be called as an RPC as it requires changing a network variable. This can only be done on the server, therefore a server RPC is necessary. The square's current piece is a network variable as it should be synced across the network at all times. This is because it is a constant state rather than a single event, so it is important for it to be consistently the same.

4. **Ending turn & timeline** – similarly to placing pieces, this will require multiple methods to implement.
   a. **EndTurn()** – this will be called when the end turn button is pressed and will call the RPCs necessary to end the turn.

```
PROCEDURE EndTurn()
      UpdateActiveBoardRpc() //spawns new active board
      ChangeTurnRpc() //changes the turn
ENDPROCEDURE
```

This will call ChangeTurnRpc() which will change the current turn stored in currentTurn, and UpdateActiveBoardRpc() which will spawn a new board and set it as the current active board.

   a. **UpdateActiveBoardRpc()** – will create a new board and move it to be right of the active board. It will set this new board to be the new active board,

```
[Rpc(SendTo.Server)]
PROCEDURE UpdateActiveBoardRpc()
      activeArrow.value.SetTurn(currentTurn) //changes the display
of the arrow to be that of the player whose turn it was

      //creates on local machine
      newBoard = Instantiate(activeBoard.value)
      newArrow = Instantiate(Arrow)
      //spawns across network
      newBoard.Spawn()
      newArrow.Spawn()


      //moves the new board to be to the right of the current active
board
      newBoard.x = activeBoard.value.x + 100
      newBoard.y = activeBoard.value.y
      //moves the new arrow to be in between the current active board and the new
board
```

```
        newArrow.x = activeBoard.value.x + 50

        newArrow.y = activeBoard.value.y


        //sets the new active board and active arrow
        activeBoard.value = newBoard
        activeArrow.value = newArrow
ENDPROCEDURE
```

This updates the activeBoard network variable to be that of the new active board. This will allow for the board to be played in in the next turn, and prevent the old board from being played in. It will also update the arrow pointing to the old active board to reflect whose turn it was, and create a new arrow pointing at the new active board.

b. **ChangeTurnRpc()** – this will swap the value of currentTurn to that of the other player number.

```
[Rpc(SendTo.Server)] //only servers can edit network variables
PROCEDURE ChangeTurnRpc()
        currentTurn = (currentTurn == 1) ? 2 : 1 //change currentTurn
from 1 to 2 or vice-versa
ENDPROCEDURE
```

If currentTurn is one, then it will be changed to two. If it is two, it will be changed to one. This will lead to the current turn being passed from one player to the other.

5. **OnTurnChange() –** this will be a method in UIManager. It is an event that will be called when the value of currentTurn is changed.

```
PROCEDURE OnTurnChange()
        turnTextDisplay = (currentTurn == 1) ? player1TurnText :
player2TurnText //changes the text to P1 if it is currently player
1's turn and P2 if it is currently player 2's
        turnPieceDisplay = (currentTurn == 1) ? player1Piece :
player2Piece //changes the piece displayed in the turn indicator to
that of the player's whose turn it is
ENDPROCEDURE
```

This is called as an event and not during EndTurn() because currentTurn is a network variable. This means that there may be a slight delay before currentTurn is updated for the client. Calling this method when currentTurn is updated ensures that it is not called too soon on the client, before currentTurn is updated, which would cause the UI to be inaccurate.