3D Connect 4

*Abstract*

*The opportunity at hand is to develop a 3D Connect 4 game in Unity, a three-dimensional adaptation of the beloved classic Connect 4 concept. The purpose of this project is to create an engaging and immersive game where two players can compete with one other online by strategically dropping their own colour blocks into a 4x4x4 grid to connect four of their blocks in a row, column, or diagonally while preventing their opponent from doing the same. It is made to promote mental dexterity within the users as it helps them think laterally within the three dimensions of the game.*

*Within the 3D connects 4 games there is going to be a multiplayer function and an AI function. This means that player who connect our community will be able to play with another existing players. Moreover, they can play against an AI to join their strategy without the risk of losing points.*

*The main core of the game will be done under Unity platform. It is a platform that gives users the ability to create games and experiences in both 2D and 3D (in our case I will be focusing on the 3D implementation and the engine offers a primary scripting API in C# using Unity main framework, Mono – these will all be discussed within analysis)*

*Additionally, games that is play between players will be able stored in a MySQL database, enabling users retrieve previously played games, thus allowing them to analyse their performance and to fine tune their strategy.*

Table of Contents

[NEA Analysis 5](#_Toc164104119)

[Background to the thought process of the project 5](#_Toc164104120)

[Questionnaire 6](#_Toc164104121)

[Frequency of Video Game Play 7](#_Toc164104122)

[Preference for Gaming Platform 8](#_Toc164104123)

[Consideration of Playing 3D Board Game 9](#_Toc164104124)

[AI Bot with varying levels in a Board Game 9](#_Toc164104125)

[Multiplayer Gameplay in a Boardgame 10](#_Toc164104126)

[Ranked Leaderboard in a Boardgame. 11](#_Toc164104127)

[Observation of Existing System 12](#_Toc164104128)

[Lichess General System Flowchart 16](#_Toc164104129)

[User Interface Analysis 17](#_Toc164104130)

[Prospective Users and Acceptable Limitations 18](#_Toc164104131)

[Research 18](#_Toc164104132)

[Game engine Analysis 18](#_Toc164104133)

[Programme Language Analysis 21](#_Toc164104134)

[MonoBehaviour Analysis 22](#_Toc164104135)

[Object Orientated Programming (OOP) 23](#_Toc164104136)

[Database Analysis 24](#_Toc164104137)

[Database Management System (DBMS) Analysis 26](#_Toc164104138)

[Server Environment Analysis 27](#_Toc164104139)

[API Request 29](#_Toc164104140)

[PHP for Server – Side Scripting 30](#_Toc164104141)

[Networking Solution 31](#_Toc164104142)

[Artificial Intelligence (AI) 37](#_Toc164104143)

[Graphics Software/ Design Tools 38](#_Toc164104144)

[Objectives 40](#_Toc164104145)

[Summary Objectives List 43](#_Toc164104146)

[Design 44](#_Toc164104147)

[IPOS Chart 44](#_Toc164104148)

[3D Connect 4 - General System Flowchart 45](#_Toc164104149)

[Registration System 47](#_Toc164104150)

[Login System 48](#_Toc164104151)

[Game Duration 50](#_Toc164104152)

[Replay Logic 53](#_Toc164104153)

[Outline Selection 56](#_Toc164104154)

[AI – Minimax Algorithm 58](#_Toc164104155)

[Network 68](#_Toc164104156)

[General Network Design 68](#_Toc164104157)

[Client Side 70](#_Toc164104158)

[Server 77](#_Toc164104159)

[UI design 102](#_Toc164104160)

[Main Menu Interface 102](#_Toc164104161)

[Login Interface 104](#_Toc164104162)

[Register Interface 105](#_Toc164104163)

[Game Interface 106](#_Toc164104164)

[Endgame Interface 108](#_Toc164104165)

[Replay Interface 109](#_Toc164104166)

[Online Interface 111](#_Toc164104167)

[AI Interface 113](#_Toc164104168)

[API 114](#_Toc164104169)

[Database 115](#_Toc164104170)

[Initial design 115](#_Toc164104171)

[Entity Attribute Models 116](#_Toc164104172)

[Final Database 118](#_Toc164104173)

[SQL Statement 118](#_Toc164104174)

[Save Game 118](#_Toc164104175)

[Register: 119](#_Toc164104176)

[Login 119](#_Toc164104177)

[Retrieve Data for Replay 119](#_Toc164104178)

[Retrieving Moves 121](#_Toc164104179)

[Data Serialisation and Deserialisation: 121](#_Toc164104180)

[UML 123](#_Toc164104181)

[Technical Solution 124](#_Toc164104182)

[Testing 125](#_Toc164104183)

[Evaluation 126](#_Toc164104184)

[Comparing performance against the objectives 126](#_Toc164104185)

[Independent feedback 134](#_Toc164104186)

[What would I do to further improve my Project 137](#_Toc164104187)

[Code Appendix 139](#_Toc164104188)

[DBManager 139](#_Toc164104189)

[GameUI 140](#_Toc164104190)

[Login 141](#_Toc164104191)

[Register 142](#_Toc164104192)

[AI Mode 144](#_Toc164104193)

[Camera Rotation 155](#_Toc164104194)

[Column Input 157](#_Toc164104195)

[Game Manager 158](#_Toc164104196)

[Outline Selection (Highlight function) 163](#_Toc164104197)

[Playfield 165](#_Toc164104198)

[Server 173](#_Toc164104199)

[Server Network Handler 188](#_Toc164104200)

[Client 191](#_Toc164104201)

[Client Network Handler 194](#_Toc164104202)

[Network Manager (Joining Games) 197](#_Toc164104203)

[NetUtility 198](#_Toc164104204)

[NetMessage 200](#_Toc164104205)

[Game Entry 209](#_Toc164104206)

[SceneManagerNavigation 216](#_Toc164104207)

[PHP/ API Request: 219](#_Toc164104208)

[Function Table 225](#_Toc164104209)

[Data Table 232](#_Toc164104210)

[References: 235](#_Toc164104211)

# NEA Analysis

## Background to the thought process of the project

Despite the enduring popularity of strategy board games like Connect 4, there has been little innovation in transforming these games to utilise the full capabilities of modern computing, particularly regarding spatial reasoning and online play. Traditional Connect 4's two-dimensional gameplay limits the strategic depth and fails to engage players in the more complex spatial thinking that three-dimensional games can offer. Additionally, existing digital versions of Connect 4 lack the community features that modern players expect, such as real-time online multiplayer, competitive matchmaking, replay function and an AI bot to play against.

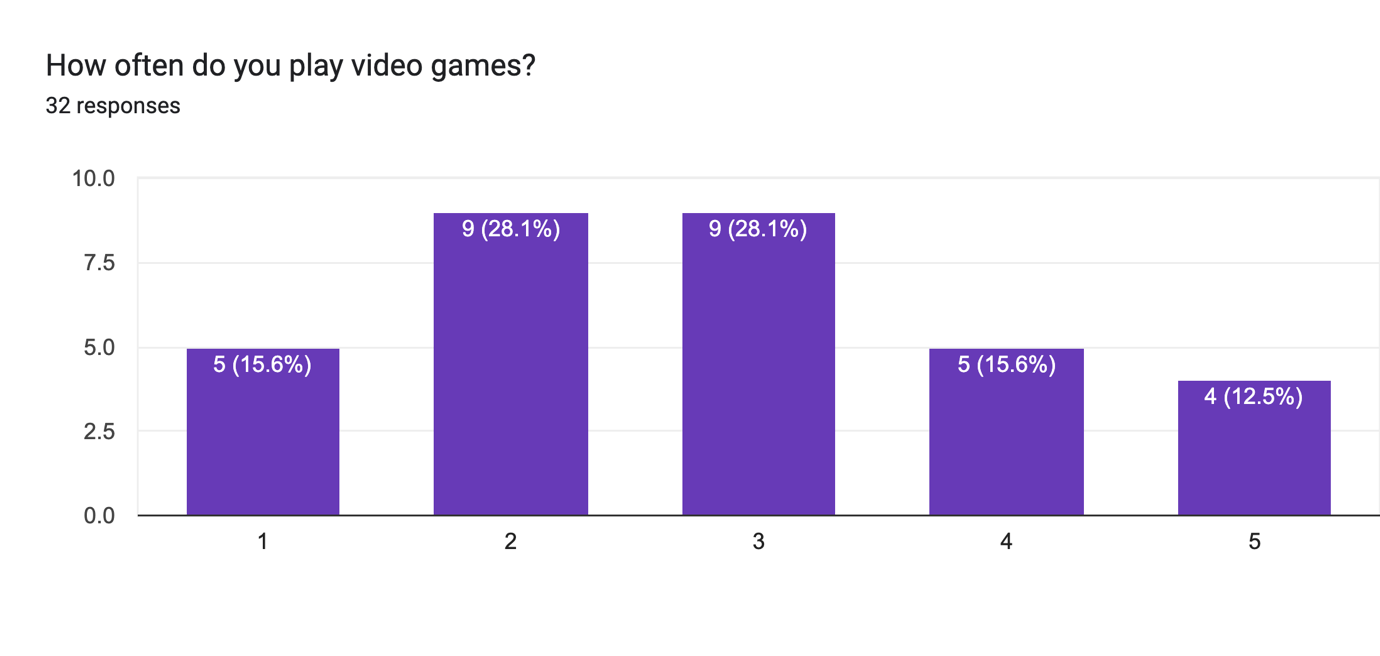
## Questionnaire

I conducted a survey on reddit to gauge the opinion of the people for video game (specific board game) and their opinion on the different functionality for the game.

Below it shows the different questions included and the reason why I added them into the survey.

|  |  |
| --- | --- |
| Question | Explanation |
| How often do you play video games? | This question helps determine the target audience's engagement level with video games. For a 3D Connect 4 game, knowing whether players are casual or hardcore can influence the game's design to ensure it captures the attention of players with varying gaming habits. |
| What platform do you prefer for board games? | Platform preference is crucial for technical planning and resource allocation. If the majority prefer PC, you will focus on compatibility and optimization for that platform. For mobile preference, you would consider touch controls and mobile-friendly UI/UX design. |
| Have you ever considered playing 3D Board games? | Interest in 3D board games can validate the demand for your 3D Connect 4 game. If there's significant interest, it suggests a market for your game and that potential players might be looking for a new twist on traditional board games. |
| How beneficial is it to have AI bot with varying levels in a board game? | A 3D Connect 4 game with AI opponents can cater to solo players or those looking to practice before playing against human opponents. Varying difficulty levels would ensure that the game remains challenging and accessible to players of all skill levels. |
| How beneficial would multiplayer gameplay for a board game be? | Multiplayer features are often key to retention and engagement in games. For your 3D Connect 4 game, knowing how players value multiplayer can guide you to prioritize features like online matchmaking, friend invites, or local multiplayer modes. |
| How beneficial do you think a Rank leaderboard is? | Leaderboards can encourage competition and provide players with a sense of achievement. In a 3D Connect 4 game, a leaderboard could add a competitive edge, encouraging players to improve their skills and play more frequently to climb the ranks. |

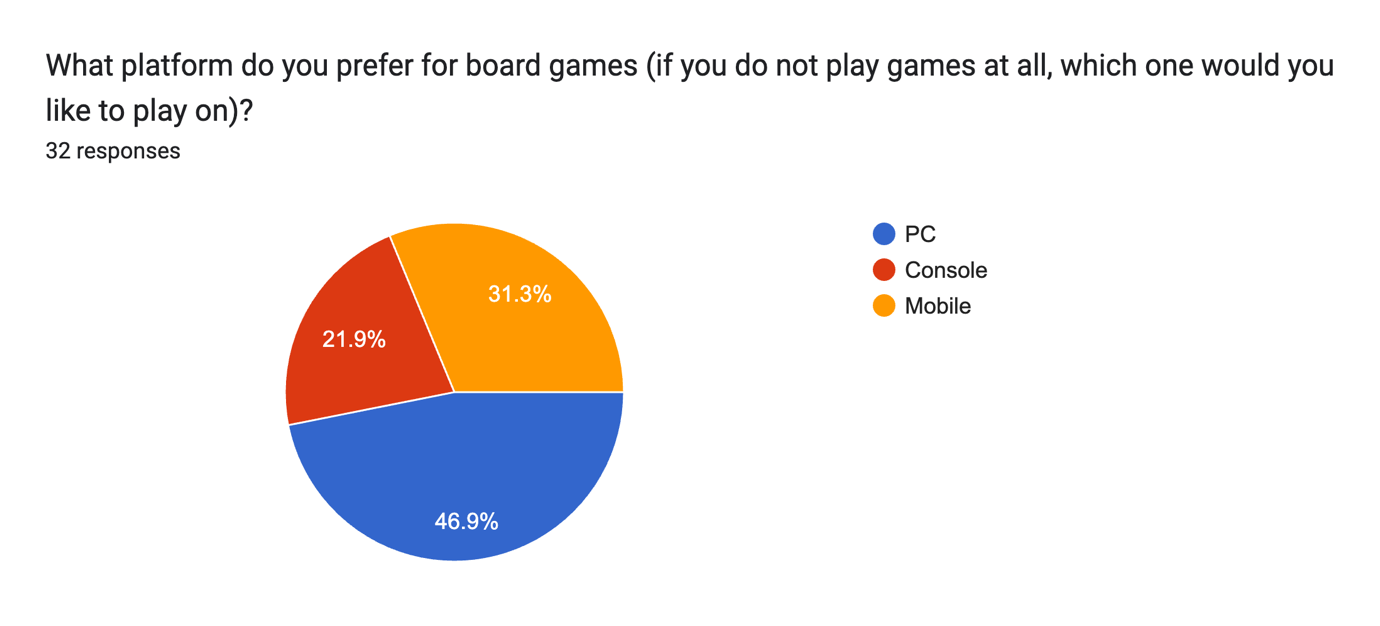
### Frequency of Video Game Play



* **Option 1 (Never)**: 5 respondents (16.5%)
* **Option 2 (Rarely)**: 9 respondents (28.1%)
* **Option 3 (Occasionally)**: 9 respondents (28.1%)
* **Option 4 (Frequently)**: 5 respondents (16.5%)
* **Option 5 (Very Frequently)**: 4 respondents (12.5%)

The first chart suggests that the respondents are fairly evenly distributed in terms of how often they play video games. The largest groups are at the extremes, with 9 respondents (28.1%) who play very frequently (rating 5) and another 9 (28.1%) who play less frequently (rating 1). This indicates a polarized audience with regards to gaming frequency, which may suggest targeting features for both heavy and casual gamers in online game development.

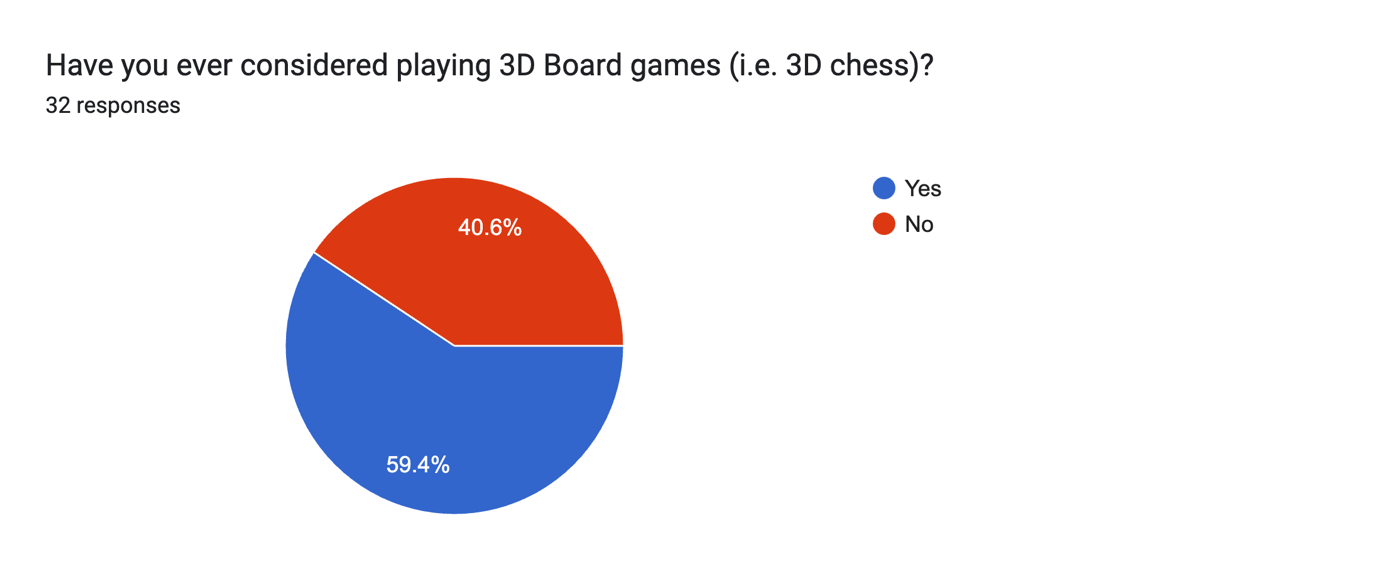
### Preference for Gaming Platform



* **PC**: 15 respondents (46.9%)
* **Mobile**: 10 respondents (31.3%)
* **Console**: 7 respondents (21.9%)

The second chart shows a clear preference for PC as the platform for playing board games, with almost half of the respondents (46.9%) choosing it. Mobile is the second preferred platform with 31.3%, while console is the least preferred with 21.9%. This could suggest focusing on PC and mobile platforms for online board game development due to their higher popularity among the survey participants.

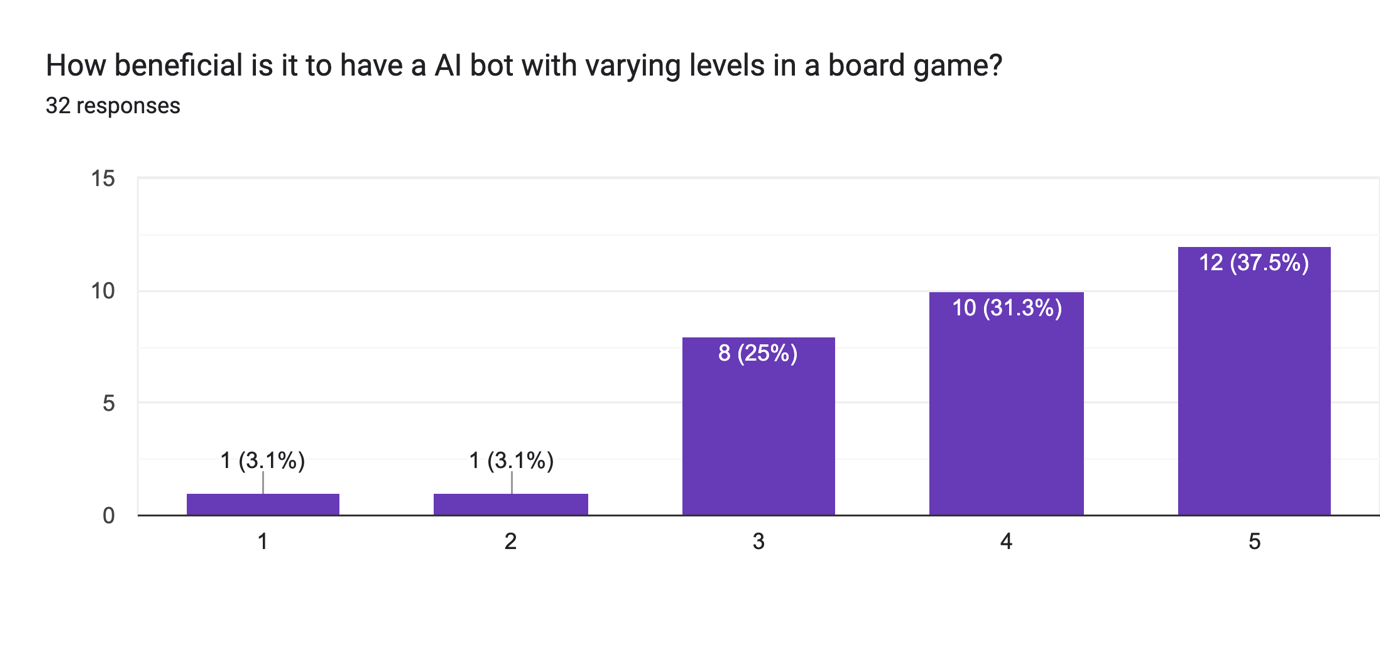
### Consideration of Playing 3D Board Game



* Yes: 13 respondents (40.6%)
* No: 19 respondents (59.4%)

The majority of the respondents, 59.4%, have considered playing 3D board games like 3D chess. This suggests a solid interest in 3D board games, which could be an encouraging sign for developing a 3D Connect 4 game, indicating potential user interest.

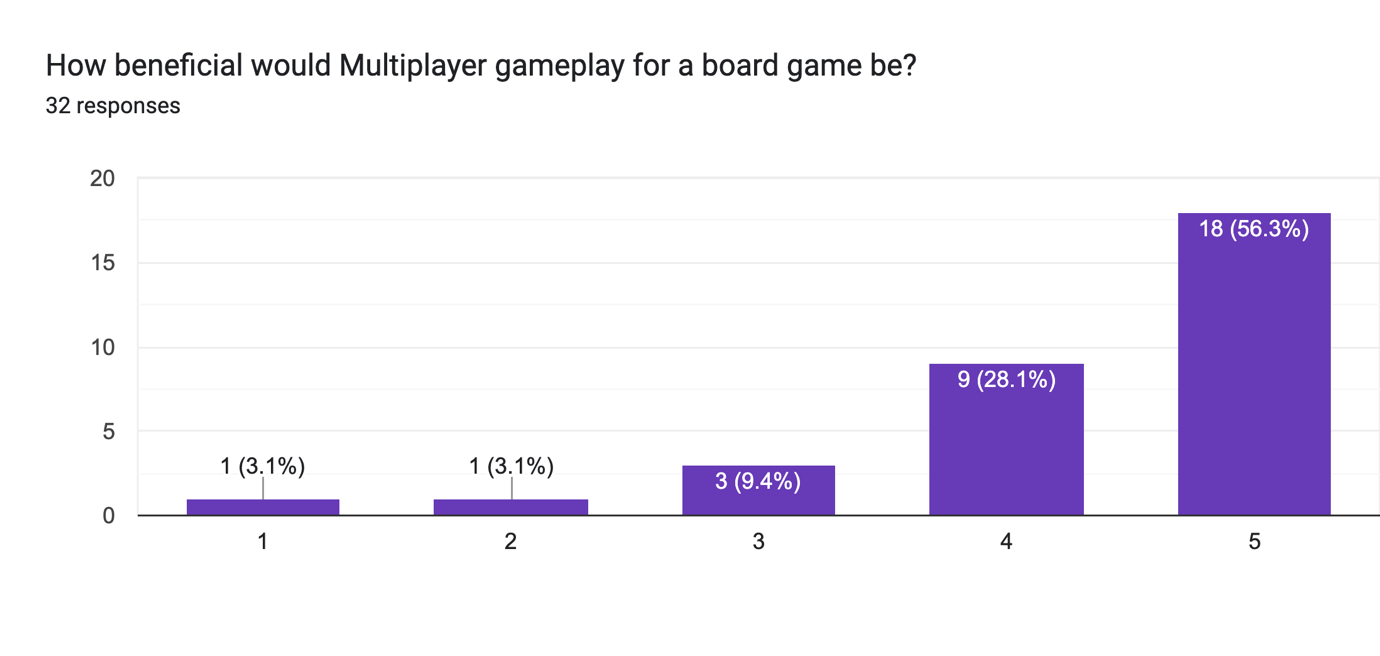
### AI Bot with varying levels in a Board Game



* Option 1 (Not Beneficial): 1 respondent (3.1%)
* Option 2: 1 respondent (3.1%)
* Option 3: 8 respondents (25%)
* Option 4: 10 respondents (31.3%)
* Option 5 (Very Beneficial): 12 respondents (37.5%)

There is a recognition of the benefits of having an AI bot with varying levels in a board game. Most respondents rated the benefit as high, with the majority (68.8%) rating it 4 or 5. This suggests that including adjustable AI difficulty could enhance the player experience, catering to both beginners and more experienced players.

### Multiplayer Gameplay in a Boardgame

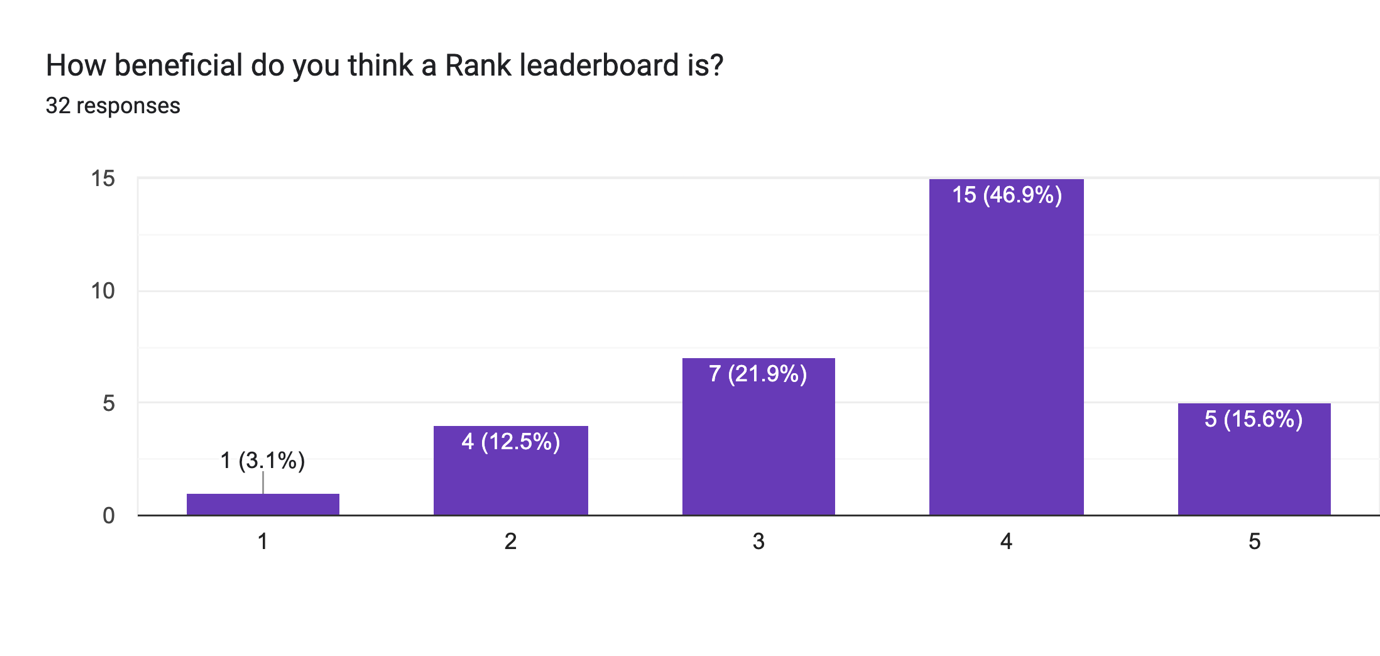


* Option 1 (Not Beneficial): 1 respondent (3.1%)
* Option 2: 1 respondent (3.1%)
* Option 3: 3 respondents (9.4%)
* Option 4: 9 respondents (28.1%)
* Option 5 (Very Beneficial): 18 respondents (56.3%)

There is a strong preference for multiplayer gameplay, with 84.4% rating it beneficial (4 or 5). It underscores the importance of social interaction in gaming, suggesting that multiplayer features should be a focal point in the game design to foster community and competitive play.

Rank Leader Board in a Boardgame

### Ranked Leaderboard in a Boardgame.



* Option 1 (Not Beneficial): 1 respondent (3.1%)
* Option 2: 4 respondents (12.5%)
* Option 3: 7 respondents (21.9%)
* Option 4: 15 respondents (46.9%)
* Option 5 (Very Beneficial): 5 respondents (15.6%)

The data shows a positive view of rank leaderboards, with 62.5% rating its benefit as a 4 or 5. Leaderboards seem to be viewed as a positive feature that could motivate players through competition and provide recognition for skill and achievement, which can be critical for engagement and retention in online games.

Conclusion:

These insights from the survey can help inform the development of the 3D Connect 4 game by highlighting the importance of features such as varying AI difficulty, multiplayer options, and competitive aspects like leaderboards. Each of these elements can add value to the game and cater to the preferences and habits of the potential player base.

Top of Form

Bottom of Form

## Observation of Existing System

Currently, there are no open – source platforms available that offer users the ability to play 3D Connect 4.

To bridge this gap, it is essential to study existing systems that parallel the user experience envisioned for our game. Lichess, an open-source chess platform, provides an ideal case for such an analysis. Its open-source nature allows for an in-depth examination of both the technical implementation and the user interaction model. Analysing Lichess will offer valuable insights into developing an intuitive and engaging user interface, establishing a robust online multiplayer framework, and understanding the nuances of maintaining a vibrant gaming community. This examination will inform the development of a 3D Connect 4 game that aims to deliver a comparable, if not superior, user experience etc.

The current process is as follow:

1. Player is processed into the menu interface.
   1. Players given multiple different options to choose from
      1. Quick Pair
      2. Sign in on the top left
      3. Puzzles
      4. Tools to analysis
      5. Learn

A screenshot of a video game

Description automatically generated

1. Players here can then branch out to various options. Here are a few examples of the various options.
   1. Quick Pair/ Play Game/ Create a Game
      1. They are then fed to the lobby page which shows you the different game that are being hosted.
      2. If you are hosting a game, then it will show up on the page.
         1. If you pressed quick join, then it will join a random game of your chosen time from the lobby.

A screenshot of a video game

Description automatically generated

* 1. If users press onto the Tools bar then they are directed to another page which has a board and there are tools on the bottom right hand side which allow the users to play forward and back

A screenshot of a computer

Description automatically generated

* 1. If user press onto the Puzzle bar then they are displayed a puzzle on the board
     1. If the users is unable to solve it then there is a “View Solution” Option on the bottom right

A screenshot of a computer game

Description automatically generated

* + 1. If they successfully solved the puzzle then a “continue training” bar will show up to the next puzzle. If they failed to solve the puzzle “continue training” will show up, but they can also reattempt the puzzle again.

A screenshot of a computer game

Description automatically generated

1. Finished Game
   1. It displays who won and reason they won.
   2. When game finishes the user is given multiple different options
      1. Rematch (the rematch function is only on you can see if the other play has not left, else it is not an option)
         1. Rematch will restart the board, but the players will switch colours – so that the order is reversed
      2. New opponent – New Match
         1. It brings the user back to the lobby like when they create/join step
      3. Analysis Board
         1. Analysis Board will bring you to the page as if you clicked on Tools
      4. Exit
         1. To exit just click on lichess.org logo to escape the game

A screenshot of a computer game

Description automatically generated

1. Sign in:
   1. It is just a very simple UI which allow you to sign in, there are other options below if you have not had a account, such as Register, password rest, log in by email

A screenshot of a computer

Description automatically generated

### Lichess General System Flowchart

In this section, I show my interpretation of how the how Lichess General System looks like, this was not provided by the Lichess website. However, I believe that it gives me a foundation to analyse from. Note: in this flowchart I did not include a registration and login processes because in every single game, it is convention to register or login, before starting, thus I did not see the purpose of adding on. In addition, in Lichess, users can play as guest, therefore, it is not a very vital part of the system – as guest get the same functionality as a logged in user.

A diagram of a company

Description automatically generated

The flowchart models the system as it is now. The flowchart Is a rough idea of what user interaction with the system is like. This enables me to see the intricate parts, allowing me to replicate the good parts and improve upon potential problems with it.

I believe that the system that Lichess is using is quite efficient and considering that they have over 100,000 players in parallel on average, suggests that their model is highly effective in handling traffic.

User Interface Analysis  
Based on the different screenshots that I have showed you of Lichess, it is easy to say that it has been well designed.

I believe that analysing Lichess will be very helpful for me to have a gauge of what is the convention design for a Turn base game.

Firstly, one key thing that stood out to me about their UI is the minimalistic design approach. I thought that the UI of Lichess is sleek and uncluttered. It uses a dark tone theme which is generally easier on the eyes and contrasting with brighter parts to highlight different elements, making them stand out for better visibility.

Secondly, Lichess also have a very much intuitive navigation system. The options are clearly laid out on the different pages. Users can easily find different game modes, access puzzles, or use analysis tools without having to be confused.

Thirdly, the accessibility of very important feature like Quick Pair, Sign In and Puzzles are prominently placed. This user centric design approach where frequently used features are the easiest to access makes Lichess’ UI very good.

## Prospective Users and Acceptable Limitations

At present, the gaming community and board game lovers will be the main users of the system. While the gaming community involves people over the age of 15, nevertheless, it is important to consider for a range of age groups. As a result, the complexity of the new system should be put into consideration when developing.

There are also other limitations as follows:

* Hardware and software constraints
* My skills and knowledge
* Time constraints
* Cost Constraints

## Research

### Game engine Analysis

The selection of an appropriate game engine is a crucial decision in the development to a game. At this day and age, there are many different game engines to choose from, for example: Unity, Unreal engine and Godot.

Unity

|  |  |
| --- | --- |
| Pros | Cons |
| **Cost** - Free for projects generating less than $100,000 per year | **Access to read** – source code is only possible with Unity Enterprise subscription |
| **Wide Adoption/ Community Support and Built-in Assets**- Unity is one of the most widely used game engines globally, with a massive and active community. This mean access to a wealth of tutorial, assets and support forums to aid in development | **Multiplayer accessibility** - Advanced multiplayer features are only available by downloading frameworks |
| **Cross – platform ability** - Coverage of more than 25 platform with an intuitive interface – enable cross – platform development in future development, reaching a broad audience |  |
| **Visual Excellence and Graphics Capabilities** - Unity offers powerful rendering capabilities, making it suitable for creating visually stunning 3D games, which aligns with the immersive experience desired for 3D Connect 4 |  |
| **Ability for Visual Scripting** –Unityoffers visual scripting, allowing the creation of game logic through a node - based system |  |
| **NET Integration –** Unity uses C# as its primary scripting language and integrates will with the .NET ecosystem, making it appealing for both beginners and experienced programmers |  |

Unreal Game Engine

|  |  |
| --- | --- |
| Pros | Cons |
| **High Rendering Performance** – it has industry – leading rendering speeds. Their graphical capabilities are far ahead of the competition with its AAA-quality visual | **Complexity and Learning Curve** – its complexity and learning curve can be steep. According to game design |
| **Animation** – powerful rendering capabilities and top-notch visual effects, its animation quality bar none. It has a robust sets of tools for advanced game development | **Difficult Programming language** – Unreal Game Engine uses C++ which requires more programming experience than C# or JavaScript |
| **Drag – and – Drop Visual Programming** - Unreal Game Engine uses a system called “Blueprint Visual Scripting”. This is a drag – and – drop programming system which enables designers to create various game aspects without writing a single line of code. | **Cost** – while the software is free, however, in the future if developer wants to release their game, Unreal Engine is entitled to royalty fee on all game sales. |
| **Wide Adoption/ Community Support and Built-in Assets –** like Unity, Unreal Game engine is one of the more widely used game engines. Similarly, it offers a wide range of Assets and Examples on the Marketplace., some of which are free | **Performance issue –** the engine builds are not optimised well for lower spec devices |

Godot

|  |  |
| --- | --- |
| Pros | Cons |
| **Multiple Programming Languages –** supports languages including C++ and Rust. It also has its custom Python-like GDScript language. | **Learning Cure for 3D** – while Godot is excellent for 2D game development, its 3D capabilities are considered even less user – friendly compared to engines like Unity or Unreal Engine. |
| **Active Community** – Godot has an active community, i.e. Godot has reddit forums, Facebook groups and steam community. This means that if user ever encounter any problems, there will always a community that can help. | **Small Community** –while does have an active community, in comparison to Unity and Unreal game engine it has a relatively small community. |
| **Cost** – it is one of the only game engines which is completely free. No royalties, no subscription fees, no hidden strings – whatever you develop through the Godot engine is yours. | **Limited Built – in Assets** – due to its nature of having a small community. This means there is fewer third – party assets, plugins and resources available |
| **Visual Programming** – Godot provide visual scripting language called “VisualScripting” which is provides a visual interface and visual elements, i.e. blocks, nodes or icons to represent logic functionality |  |
| **Multi – platform ability –** Godot is known for its strong cross – platform capabilities when it comes to game development. |  |

After evaluating the various game engines, Unity emerged as the optimal choice for our 3D connect 4 projects. The decision was based on several critical factors, including graphic, capabilities, ease of use, project backup ability and programming language used. However, it is essential to acknowledge that other game engines also have their strengths and weaknesses.

### Programme Language Analysis

C# for Unity

Since we have decided to use Unity, it is important for to acknowledge the benefits and drawback of C# as it is predominant programming language for Unity.

C# is an objected – oriented programming language which is developed by Microsoft. It is the primary programming language supported by Unity. C# is a popular choice among Unity Developers because of its strong typing, performance and the extensive .NET framework library. The .NET framework provides a vast class Library which provides a wide range of pre – built classes and functions for common tasks, i.e. networking, data access and user interface development.

In Unity, C# scripts are used to define how game objects in the game behave. C# scripts usually contain functions and logic that controls the game. Users are not only able to access the built-in components and classes, in addition, users are able to create their custom classes and scripts to extend the functionality of your game.

|  |  |
| --- | --- |
| Pros | Cons |
| Rich standard Library – .NET framework means that users can access a vast amount of pre – built classes and functions. This saves time for users and | Overwhelming for newcomers – in comparison to other languages, such Python and JavaScript, C# can be slightly overwhelming for new programmers. Its Object – Oriented Programming principles can be complex for beginners. |
| Cross – Platform - with its .NET Core, C# is increasingly cross – platform. You can develop and run C# on Windows, macOS and Linux |  |
| Large and Active Community and Support – C# has a large and active developer community, which means there are abundant resources, documentation, and online forums for learning and troubleshooting. |  |

### MonoBehaviour Analysis

Why is MonoBehaviour important?

MonoBehaviour is a fundamental class in Unity engine that serves as the base class for almost all scripts that is going to be created in this game. It is crucial to understand MonoBehaviour for Unity development, as it provides the framework for interacting with the unity editor and the game lifecycle itself.

Concept:

MonoBehaviour is part of the unity Engine namespace ‘UnityEngine’. It enables the scripts to communicate with the Unity editor and the lifecycle of game objects. Scripts derived from MonoBehaviour can override specific methods which Unity calls at various points in a game object’s lifetime, such as when the game starts, when the game updates, or when the object is destroyed.

Lifecycle and Callbacks:

One of the key features of Monobehaviour is its event functions that are automatically called by Unity at specific points.

Example of callbacks:

|  |  |
| --- | --- |
| Method | Description |
| Awake() | Called when the scripts instance is being loaded |
| Start() | Called before the first frame update, after all ‘Awake’ functions have been called. This is where variables and game state get initialised |
| Update() | Called once per frame. Ideal spot to do any listening for input and updating game logic |
| FixedUpdate() | Called at a fixed frequency, making it ideal for physics related updates |
| DontDestroyOnLoad() | Do not destroy the target Object when loading a new Scene. Used very often when a script still have to be used in a different scene |
| Destroy() | Removes a GameObject, component or asset. |

Usage in the Unity Editor:

MonoBehaviour scripts are attached to GameObjects within the Unity Editor. This association allows different scripts to control various game object’s behaviour, appearance, and interactions within the duration of a game.

Conclusion:

MonoBehaviour is a huge part of Unity development, providing a structured way to implement game behaviour and interactions. Understanding its lifecycle, how it interacts with the Unity Editor, and how to use it effectively is essential for developing my 3D Connect 4 game in Unity. Through MonoBehaviour, I am able to script the core mechanics and interactions needed for the game to work accordingly.

### Object Orientated Programming (OOP)

Analysing the principles of Object-Oriented Programming (OOP) is crucial when using C# within Unity for the development of my 3D Connect 4 game. The choice of programming paradigm significantly influences the code's efficiency and adaptability, matching Unity's inherently object-oriented architecture. A closer examination of OOP’s fundamental aspects—encapsulation, inheritance, polymorphism, and abstraction—enables us to harness the full potential of C# to construct a robust and expandable game framework.

Encapsulation in C# promotes a modular design by allowing game components to be constructed as objects with their unique data and behaviours. This approach not only streamlines the code, making it more manageable but is also invaluable for crafting complex game logic, such as that required for 3D checks and validations in Connect 4.

With inheritance, C# facilitates a hierarchy of classes capable of acquiring methods and properties from others. This structure encourages common behaviours to be defined once and reused across various game objects, enhancing code efficiency and diminishing repetition.

Polymorphism further refines OOP's capabilities, granting game objects the power to supersede inherited methods and adopt behaviours tailored to their role, essential for custom actions such as player moves and AI responses (which will be explored further later). This feature is particularly advantageous in a gaming context where varying game piece types and rules necessitate distinct behaviours.

In summary, the OOP features of C# form a formidable toolkit for building the intricate components of a 3D Connect 4 game. Adopting OOP not only streamlines the management of game components but also underpins the game's core logic.

### Database Analysis

Selecting the appropriate database is a critical decision in the architecture of 3D Connect 4, as it lays the foundation for the backend infrastructure that will drive the game's functionality.

I will be focusing on analysing SQL (Structured Query Language) and NoSQL (Not Only Structured Query Language) as they are the two most used method for games.

|  |  |  |
| --- | --- | --- |
| Features | SQL | NoSQL |
| Query Language | Declarative SQL language | No declarative query language varies by NoSQL type (eg MongoDB uses JSON-like query language) |
| Data Structure | Structured data with predefined schema | Dynamic schema for unstructured data |
| Scalability | Vertical Scaling; complex horizontal scaling | Horizontal scaling |
| Transactions | ACID compliant transactions  (Atomicity, Consistency, Isolation, Durability) | BASE compliant transactions  (Basically Available, Soft State, Eventually Consistent)  ACID does not guarantee |
| Complexity of Queries | Complex queries with joins | Simpler queries: joins are less common or more complex to implement |
| Schema Modification | Rigid; modification can be complex | Flexible; easy to modify and extend |
| Consistency | Strong (ACID principle) | Eventual (BASE principle) |
| Data Model | Relational (mainly RDBMS with SQL) | Key-value, document, wide-column, graph |
| Performance | Optimised for complex transactions | Optimised for speed and large data volume |
| Community and Support | Extensive and mature | Growing, with a focus on modern application |

After consideration of the game’s data management requirements, SQL has been chosen over NoSQL for the 3D Connect 4 game.

The decision is grounded in several key factors:

* **Structured Data**: Our game data is organised and consistent, perfect for SQL's table-based format.
* **Complex Queries**: We need the ability to perform complex data operations, which SQL handles well.
* **Reliability**: Game updates and transactions are critical and must be handled reliably, something that SQL's ACID compliance ensures.
* **Data Relationships**: The game's data elements are interrelated, requiring the relational capabilities of SQL.
* **Proven Stability**: SQL's established history provides a broad support network, making it a safe choice for our game's backend.
* **Time**: One of the limitations of this project is time and since I have already previous have experience with various SQL databases, it means that I will not need to spend more time on learning.

**Consider during development:**

While SQL databases traditionally rely on vertical scaling (scaling up), there are a few solutions managing horizontally scaling.

Examples of solutions for SQL:

Database Sharding:

Data is split into several nodes based on a shard key. Each shard contains a subset of data, making it faster to manage data across all shards. Sharding improves both read and write performance of the data store since each database is handling lower volumes of data.

Shards 
Database Sharding 
Cust A-D 
Cust K-N 
Cust E-J 
Cust O-S 
Cust T-Z 

Database Caching:

Keep frequently accessed data in an in-memory cache and save the trip to the database.

To fetch the data from the database you need to execute a stored procedure or query involving multiple tables, which might be an expensive operation. Instead of making this database call every time, you can store the value in the cache, and the next time this data is required, you can return it with sub-millisecond latency.

A green square with white text

Description automatically generated

### Database Management System (DBMS) Analysis

After considering using SQL database for my game, I now need to choose the DBMS to use. This part is mainly personal preference. However, I believe it is still important to do a comparison table for the most commonly used SQL database to see which one is the most suitable.

**Comparison Table of SQL DBMS Options**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Features | MySQL | PostgreSQL | MariaSQL | SQLite |
| Popularity | High | High | Medium | High |
| Ease of Use | Easy | Medium | Easy | Easy |
| Performance | Good | Excellent | Good | Moderate |
| Scalability | Moderate to High | High | Moderate to High | Low |
| Concurrency | Good | Excellent | Good | Moderate |
| Security | Good | Excellent | Good | Moderate |
| Cost | Free (Open Source) | Free (Open Source) | Free (Open Source) | Free (Open Source) |
| Extensibility | Good | Excellent | Moderate | Moderate |
| Community | Very Large | Large | Moderate | Large |

After a comprehensive analysis of various SQL DBMS options, MySQL has been chosen as the database management system for the 3D Connect 4 game project. MySQL strikes a balance between ease of use, performance, and scalability, making it an attractive option for a game expected to grow in user base and complexity. Its robust transaction management capabilities ensure that game state updates and player interactions are handled reliably, which is crucial for maintaining the integrity of multiplayer sessions.

Furthermore, MySQL’s extensive popularity translates into a vast community and a wealth of resources, which is invaluable for ongoing support and learning. The open-source nature of MySQL not only aligns with budget considerations but also offers the flexibility to customise and scale the database according to the evolving needs of the game. However, with it being open source, there is also slight drawback in terms of privacy and security. This can be resulted due to yet to reported bugs being exploited causing security and privacy issues.

In conclusion, its compatibility with various platforms and technologies used in game development ensures seamless integration. All these factors, combined with prior experience with MySQL, form a strong foundation for its selection as the DBMS for the 3D Connect 4 game.

### Server Environment Analysis

A Server Environment is used during the development stage for developers to host a local server on user’s computer. It is an open – source database server management package that allows users to manage the server. In this analysis, our aim is to find a one that can create a SQL database for our game to store user information (i.e. login, password, points) and games.

XAMPP (Cross Platform, Apache Server, MariaDB, PHP and Pearl)

|  |  |
| --- | --- |
| Pros | Cons |
| Cost – it is a free open-source platform as a result this makes it a free local server environment that can be installed by anyone | Management issues - Developers find it difficult to manage configuration and other settings as compared to the WAMP stack model |
| Ease to set up – compared to other stack packages such as WAMP, XAMPP is much easier to install | Only MariaDB support – XAMPP uses MariaDB as their default database system, which is an alternative to MySQL. |
| Cross – platform capability - XAMPP is available for multiple Operating System, including Windows, MacOS, and Linux, making it versatile choice for web development | Database Security - XAMPP is not configured for the security measures necessary for protecting databases in production environment. |

MAMP (Macintosh, Apache Server, MariaDB, PHP and Pearl)

|  |  |
| --- | --- |
| Pros | Cons |
| Ease of Setup (macOS Focus)/ Quick Installation – MAMP is designed specifically for macOS, making it a convenient choice for developers working exclusively on Apple computers. Its user – friendly one – click installer and interface are tailor to macOS users. This straightforward installation process, means it can be up and running within minutes | Database Security - MAMP is not configured for the security measures necessary for protecting databases in production environment. |
| MySQL and MariaDB Compatibility – MAMP’s uses MySQL and MariaDB for data storage and retrieving operations, in comparison to XAMPP which is only limited to MariaDB for database. This enables versatility for users depending on their liking | Cost – MAMP offers both free and paid version. With its pro version, developers can create separate host for each web project. |
| Multiple Versions of PHP – MAMP allows easy switch between different versions of PHP, which is useful if the project requires compatibility with specific PHP versions. |  |
| PHPAdmin – MAMP included PHPAdmin GUI for managing MySQL databases, making it convenient for database - related tasks. |  |

Conclusion:

There are many similarities between MAMP and XAMPP. Firstly, the Cost of both stack package are cost effective options as they offer free versions for local development environment. Secondly, both provides a relatively straightforward installation process. Lastly, both provides phpMyAdmin which allows user to manage and interact with the database.

However, at the end, it has emerged that MAMP will be more beneficial for the project. The is decision to choose MAMP was based on 2 key factors.

MAMP is Mac – Centric making it ideal for this project, given that it will be exclusively carried out on a Macintosh device. Unlike XAMPP, which is designed to function across different platforms, MAMP is tailored specifically for macOS users, simplifying the setup process and ensuring seamless integration.

Secondly, my prior experience with MySQL, which is the database system used by MAMP, played a significant role in the decision – making process. Unlike XAMPP, which employs MariaDB for data storage and retrieval operations, MAMP’s use of MySQL aligns perfectly with my existing knowledge. This familiarity hopefully allows a smoother developing experience when working with database.

It is important to remember this is not the end solution for the game’s server, it is only going to be used in the development stage of the game since the security of such server is only as secure as the user’s computer. In the future if the game wanted to be launched for the public, then the database would be migrated to an official cloud server (ie AWS) which provides greater security. However, keeping in mind with cost as one of the limitations, we will use MAMP for the moment.

### API Request

Unity does not inherently support direct database interactions like web development framework might. Therefore, to retrieve or manipulate data stored in a database, I would need to set up a server-side application that Unity can communicate with This is where an API comes into play, allowing Unity to communicate with the database.

Here is the API role in the project:

1. **Database Interactions**: Unity itself doesn't have built-in support for directly interacting with databases like MySQL. To store and retrieve game data (like player scores, game states, or replay moves), I need a way to safely and effectively communicate with a database. This is where the API comes into play.

Reason for API interaction:

1. **Separation of Concerns:** By using an API, you separate the game logic and database management. This approach follows the principle of separation of concerns, making my project more organized and manageable. The Unity application focuses on the frontend and game dynamics, while the API handles data persistence, retrieval, and other backend tasks.
2. **Security:** Directly accessing a database from a client application (like a game) exposes it to significant security risks, including SQL injection and unauthorized access. An API acts as a secure intermediary, where I can implement authentication, validation, and sanitization of inputs before they interact with the database. This setup helps protect sensitive data and ensures that only authorized requests can perform operations on the database.
3. **Maintenance:** As my game grows in complexity and user base, handling data operations directly within Unity can become unwieldy. An external API allows me to offload these tasks, making it easier to scale and maintain my application. I can update my database structure or the logic for data access without needing to redistribute or update the client application.

How it works:

1. **Sending Requests**: Within the game, when specific actions occur (like starting a new game, ending a game, or requesting a game replay), Unity sends an HTTP request to the PHP scripts hosted on my server. This request might include data necessary for the server-side logic to process the request, such as session IDs, player actions, or authentication tokens.
2. **Processing Requests**: The PHP scripts on the server act as the API endpoints. They take the request, process it (e.g., validate input, interact with the MySQL database), and then prepare a response. This processing might involve retrieving game history, saving a new game state, or calculating player scores.
3. **Responding to Requests**: After processing the request, the server sends a response back to the Unity client. This response usually contains the data requested by the Unity game or a status message indicating the outcome of the operation. The data is often formatted in JSON, which Unity can easily parse and use to update the game state on the client side.

### PHP for Server – Side Scripting

PHP scripts use for the API request and to do the SQL queries to interact with the MySQL database to perform tasks like inserting, updating, deleting and querying data.

|  |  |
| --- | --- |
| Pros | Cons |
| Easy to set up and learn – getting up and running with PHP is simple. Moreover, it is easy for beginners to pick up and start coding | Lack of Built-In Features – some modern web development features are not well – supported in PHP without additional libraries or extension |
| Large community and support – PHP community is extremely strong. PHP community is one of the most active communities. There are over 70,000 open-source libraries available | Limited Debugging tools – PHP handles errors poorly in comparison to other scripting languages |
| Cost – it is free to use, since it is a open source |  |
| Adaptable with multiple platforms – It is high flexible and adaptable. It allows PHP and MySQL to work together (this is key in the development of 3D connect 4 database) |  |

### Networking Solution

#### Network Model

There are many different Networking Architectures that can be selected for this Game. Our aim is to provide multiplayer ability and matchmaking ability for the user so that they can play against other clients.

**1. Peer-to-Peer (Client as Host)**

**Description:** In this model, one of the clients in the game session also acts as the server. This client is responsible for managing the game state and communications between all players.

**Pros:**

* **Reduced Infrastructure Costs**: No need for a dedicated central server, reducing overhead costs.
* **Simpler Setup**: Can be easier to set up for small-scale or local multiplayer games.

**Cons:**

* **Dependence on Host's Connection**: The quality of the game can be heavily dependent on the host's internet connection and hardware.
* **Limited Scalability (with consideration of over reliant on Clients’ devices processing power)**: Not suitable for large numbers of players; can struggle with high latency and synchronisation issues. It relies too heavily on the Clients’ computer’s processing power in terms of the performance of the gameplay.

**2. Traditional Peer-to-Peer (No Central Host)**

**Description:** In this pure P2P model, there is no central host. Each client communicates directly with all other clients, and each one independently manages a part of the game state.

**Pros:**

* **Decentralisation**: Eliminates the need for a central server, potentially reducing latency between players.
* **Resilience**: More resilient to single points of failure since there is no central server.

**Cons:**

* **Complexity in Synchronisation**: Ensuring all players have a consistent view of the game state can be challenging.
* **Security Risks**: More susceptible to cheating and harder to implement effective security measures.

**3. Server Client-Server (C2S)**

**Description:** A dedicated server, separate from any of the clients, manages all aspects of the game. Clients connect to this server to play the game.

**Pros:**

* **Consistent Performance**: Offers stable and consistent game performance, as the server is dedicated solely to the game.
* **Security**: implement security against cheating.
* **Control**: Offers control over the gaming environment and game states of each room
* **Scalability**: Can handle a large number of concurrent players effectively.

**Cons:**

* **Infrastructure Management**: Requires managing and maintaining server hardware and software.

Conclusion:

I have chosen to use a client-server architecture. A client-server algins well with the goal of my 3D-Connect-4 game. This is because within the project, I want to be able to minimise number of processes being done on a client’s computer. Furthermore, I believe that with a client-server architecture, it will make future updates and patches easier to be done in comparison to the other architectures.

While a client-server model might require a more substantial initial setup and design, the benefits to the player experience outweigh these preliminary efforts. This model allows me to have control over the game logic, which allows for more effective anti-cheat measures. This is because server has authority over game decisions and can validate actions from clients, making it harder for players to manipulate the game state. Furthermore, it enables an easy implementation of matchmaking services, unlike peer-to-peer networks. This greatly enhances the user experience by matching players of equal skills. Thus, I have chosen to select Client-Server model.

#### Thin v Thick Client:

After selecting a client-server network model, I decided on whether the game should have a thick or thin client approach. Below, I will compare the performance, security and maintenance and update of the two different approach and see which one aligns more to the game needs.

|  |  |  |
| --- | --- | --- |
|  | Thin | Thick |
| Description | The server will handle the majority of the game logic, including validating moves, maintaining the game state, and determining win conditions. The client would primarily be responsible for displaying the game state and capturing user input, which it then gets sent to the server for processing | Much of the game’s logic will be handled on the client side. The server’s role would be relegated to relaying messages between clients, authenticating users, and possibly storing game outcomes |
| Performance | The game’s responsiveness may heavily depend on the network quality, as each move needs to be sent to the server for validation before being reflected on both player’s screens. This could introduce latency in game updates potentially affecting the user experience in real – time gameplay. | Offers potentially better performance with reduced latency for game actions, as moves do not need to be validated by the server in real time. However, this requires ensuring both clients are perfectly synchronised to maintain game integrity. |
| Maintenance and Updates | Easier to manage as update to the game logic and features would primarily occur on the server side. Players will not need to download updates manually, ensuring all users are always on the latest version | Requires more effort to update, as changes in game logic or features need to rolled out to all clients. Ensuring compatibility across versions can be challenging |
| Security | Centralises the game logic and critical data on the server, reducing the risk of cheating and ensuring consistent game rules enforcement | Higher risk of cheating or malfunctioning of game, since the game logic resides on the client. Implementing robust security measures to prevent tampering becomes crucial |

Game Specific consideration when choosing Thin or Thick Client

* Real – time Interactivity: Some games such as CSGO are very fast paced, and the low latency is important, this is where a thick client approach might offer a smoother experience.
* Game Complexity and Updates: For a game that may update a lot, the thin client model allows for more straightforward updates without requiring players to download new versions.

After consideration, I believe that a thin client approach will be more beneficial for the game. My conclusion on a thin client model was grounded in the performance and security of the game, despite potential concerns regarding network latency, with a thin client model, it ensures that there will be no cheating or manipulation of the board as all the validation check are done on the server side. Secondly, with a thin client approach, the game does not become over reliant on the client’s computer performance as most of the game logic is done on the server. This means that less amount of people is unable to play the game due to the restriction of the their computer’s quality.

#### Application use to implement Client-Server Network:

There are many different options to choose from in order to implement Client-Server Network model. However, below I am going to show you the most frequently used applications that are used to build the network.

1. **Unity Gaming Services - Game server hosting:**

A High-Level package which provides suite of services built by Unity that included matchmaking, multi – play for dedicated game server hosting. It’s designed to help developers manage and scale their game’s backend infrastructure.

**Pros:**

* **Fully Integrated**: Unity Game Services is built by Unity. Thus, it has been well integrated into the Unity ecosystem, simplifying the development process.
* **Scalability**: Designed for scalability, it can handle projects of great size – even capable to handle large AAA titles game.
* **Matchmaking services**: Includes prebuilt matchmaking services.
* **Support and Maintenance**: Backed by Unity, it offers professional support and regular updates. This means that developers don’t have to manage their own servers.

**Cons:**

* **Cost**: all these services come in a premium price, which is most likely to be more expensive than other solutions.
* **Less Control**: Since it’s a managed service by Unity, there is a restriction on what can be done.

1. **Photon:**

A popular third-party networking framework and cloud service for building scalable multiplayer games.

**Pros:**

* **Ease of Use**:Photon is known for its user – friendly approach, with plenty of documentation and community support.
* **Cross – Platform Play**: It supports cross – platform play out of the box, which is great for games on multiple platforms.
* **Server Infrastructure**: Photon handles the server infrastructure, so developers don’t need to manage their own servers.

**Cons:**

* **Cost at Scale**: While Photon has a free tier, costs can increase significantly as the game scales up in player count and server demand.
* **Less Control**: Since it’s a managed service, there’s less control over the server compares to hosting my own.

1. **Mirror:**

A high-level networking framework for Unity, which is community driven project based on Unity’s now deprecated UNet networking system. Mirror is designed to be simple to use whole still providing a powerful framework for networking.

**Pros:**

* **Free and Open Source**: Mirror is free to use and its open – source nature allows for extensive customisation.
* **Community Support**: Has a strong community for support and is regularly updated by contributors.
* **Simplicity**: Easier to get started with compared lower – level API

**Cons:**

* **Less Comprehensive**: Doesn’t provide a full backend solution like Unity Services and Photon, so additional tools may be needed.

1. **Unity transport Layer:**

A Low-Level network API provided by Unity, which is part of the Unity Networking system. It offers more control and is designed for users who want to work closer to the network layer for highly customised networking needs.

**Pros:**

* **Control**: Offers more control over the networking which is beneficial for custom network solutions.
* **Performance**: Potentially greater performance since it’s a lower – level API and can be optimised for specific use cases.

**Cons:**

* **Complexity**: More complex to use and requires a good understanding of networking principles.
* **More Development Required**: features will have to be implemented by myself as there are no prebuilt features.

**Conclusion:**

The primary constraint in deciding the application for this project is budget. As we do not have funds allocated for a dedicated server, I was more inclined to opt for solution that minimises financial cost. This excludes options that requires a premium for usage.

Furthermore, the ability to tailor the network specifically for my game is crucial. A one-size-fits-all approach often in general services can lead to suboptimal performance for specific game mechanics. Therefore, the chosen networking solution must offer me the flexibility for full customisation, allowing me to fine-tune the network characteristics to be perfectly aligned with my 3D Connect 4 game.

#### NetMessages:

After deciding on using Unity Transport Layer to implement a client-server model, as a result, I will analysis the NetMessages standard protocol which is used with Unity Transport Layer. This protocol allows the client to communicate with the server.

The reason why I am only analysing NetMessages is because this is the only option that is used alongside of the Unity Transport library, in order to provide a tailored network for the project.

**Purpose of NetMessage and Its Role in Networked Games**

1. **Communication Protocol Foundation**: **NetMessage** serves as the foundational class for all network communications in my game. It is designed to abstract the common functionalities needed for sending and receiving data over the network, ensuring that different types of messages (e.g., game state updates, player actions, matchmaking information) can be handled efficiently and consistently.
2. **Data Serialisation and Deserialisation**: A crucial aspect of **NetMessage** is its role in data serialisation and deserialisation. This involves converting game data (like player moves, game state changes, etc.) into a format that can be transmitted over the network and then converting it back into a usable form on the receiving end. This process is essential for ensuring that game data is accurately shared between the server and clients, maintaining game synchronisation.
3. **Message Typing and Handling**: The **NetMessage** class and its derivatives typically implement a way to distinguish between different types of messages. This can be done through an ID system or type indicators within the message structure itself. By identifying the type of message, the game's networking code can route the message to the appropriate handler function, ensuring that each piece of data is processed in the context it was intended for.
4. **Efficiency and Scalability**: In networked games, especially those that may operate in real-time or near-real-time, the efficiency of message processing is paramount. **NetMessage** structures are often designed to be as lightweight as possible, containing only the necessary information for the specific action or update they represent. This minimises bandwidth usage and reduces latency, contributing to a smoother and more responsive gaming experience for players.
5. **Extensibility**: By using a base **NetMessage** class with various derived classes for specific message types, my game's networking architecture becomes highly extensible. This means that as my game evolves or requires new features, I can introduce new message types without overhauling the existing communication infrastructure. This design pattern supports modular development and simplifies the process of updating and maintaining the game.

### Artificial Intelligence (AI)

Implementation of an AI will be to increase the single experience. The AI purpose is to provide a challenging yet beatable adversary for solo players and to serve as a practice opponent without sacrificing any points in multiplayer games.

**Consideration when finding a algorithm:**

1. **Complexity Handling**: The AI must be capable of navigating a 3D space, which is inherently more complex than 2D due to the additional dimension.
2. **Strategic Depth**: It should exhibit strategic depth, understanding both offensive and defensive manoeuvres, and be able to predict player moves to some extent.
3. **Adaptive Difficulty Levels**: The AI should cater to various skill levels, from beginner to advanced, adjusting its difficulty based on the player’s performance.

**Potential Algorithms:**

1. **Minimax Algorithm**: This is a standard algorithm for turn-based games, where the AI evaluates possible moves and their outcomes.
2. **Monte Carlo Tree Search (MCTS)**: A more sophisticated approach that uses randomness in simulating multiple game outcomes. It’s particularly effective in games with a high degree of complexity.
3. **Machine Learning Approaches**: Implementing machine learning, especially reinforcement learning, can make the AI adapt and learn from each game, improving its strategy over time.

**Conclusion:**

I believe that the Minimax algorithm is the most suitable algorithm to implement in comparison to Monte Carlo Tree Search and Machine Learning Approaches.

Firstly, Minimax offers a clear and deterministic framework for decision-making that is relatively straightforward to implement and debug. This is particularly valuable in an educational setting where the primary objective is to demonstrate understanding and application of AI principles within the confines of the project's scope and timeframe.

Secondly, while Machine Learning approaches, especially reinforcement learning, have the potential to create highly sophisticated and adaptive AI players, they also require a significant investment in terms of data collection, training time, and computational resources. For an NEA project, these requirements may exceed the available resources and could complicate the project's completion within the academic schedule.

Lastly, Minimax has a predictable computational cost, which can be controlled by adjusting the depth of search. With this project, one of the limitations was computer resources. The Minimax algorithm makes it more suitable for environments where resources are limited than the other algorithm, as is often the case in educational settings.

### Graphics Software/ Design Tools

In this analysis, my objective is to identify a graphics software system with a high degree of user – friendliness, particularly since I have not had any prior experience in 3D modelling. My aim is to discover a software platform with an intuitive interface that minimises complexity, ensuring accessibility for individual at all skill levels.

Furthermore, we are looking for a software system that has a large and active community that has a wealth of tutorials and online forums. This is crucial as this will provide assistance and solution in cases of any encountered issues during the development process.

Therefore, our goals are to objectively assess and identify the most suitable software that aligns with these criteria.

In our project, our aim is to design and create the blocks and board for my 3D Connect 4 game in Unity. While there is a wide range of different software system out there, in this analysis we are going to review through the two most popular options for this task which are Blender and Autodesk maya.

Blender

|  |  |
| --- | --- |
| Pros | Cons |
| Cost – Blender is open-source software, which has a General Public licence, thus making it free to use | Specialised Tools – While Blender is comprehensive, some of Blender’s tools and features might not be as advanced or specialised as those found in Maya. |
| Active Community and Resources – Blender has a dedicated and active community, offering tutorials, add – on, and support forums that can assist you in your 3D modelling endeavours |  |
| Versatile 3D Modelling – Blender offers wide range |  |
| Learning Curve – in comparison to other graphic software, such as Autodesk Maya, the learning curve of Blender is much more comprehensible for newcomers |  |
| Constant Update – due to its active community, this means that there are always new updates which improves user’s experience |  |

Autodesk Maya

|  |  |
| --- | --- |
| Pros | Cons |
| Industry Standard - Autodesk Maya is widely used in the 3D game development industry. | Cost - Autodesk Maya is not free and comes with licensing fees, which may not be suitable for developers on a tight budget |
| Robust 3D Modelling – Maya offers a wide set of sculpting tools, ideal for creating objects to your liking | Resource Intensive – Maya requires a powerful computer for smooth operation |
|  | Learning Curve – Maya has a significant learning curve, and newcomers to 3D modelling may find it overwhelming |

In conclusion, Blender has emerged as the more suitable option for several factors. Firstly, the cost factor sets Blender apart from Autodesk Maya. While Maya requires users to pay licensing fees, Blender is open – source software accessible to anyone for free. This financial advantage makes Blender an attractive choice for users seeking a cost – effective solution since I am not funded for the project.

Secondly, Blender has an active community and there is an abundant resource online. Despite Maya’s status as an industry standard, since users have pay to use the software limits the size and scalability of its user base and community. This also results in fewer resources for learning. In contrast to Blender which has a large, active community which ensure that there is a vast number of tutorials and resources for new users.

Lastly, Blender’s learning curve is much more comprehensible for new users. This is very much a curial factor for me, who does not have any prior experience in 3D modelling. In the context of a year – long project, the ability to quickly learn how to use the basics of the software is very important. This, in turn, enables me to allocate more time and attention to other aspect of the game (I.e. building the UI, the game and the database).

# Objectives

|  |  |  |  |
| --- | --- | --- | --- |
| No | Objective | Justification | Performance Criteria |
| 1 | Design the 3D Connect 4 game board logic for a 4x4x4 grid. | Precise board logic is critical for ensuring valid game state and rules enforcement. | The logic must accurately track each move the board state, validate the legality of moves, and enforce game rules. |
| 2 | Develop the graphical user interface for the 4x4x4 3D Board. | A visually engaging interface is essential for an immersive gaming experience. | The GUI should clearly display the grid and differentiate between empty and filled spaces with high contrasting visual appeal. |
| 3 | Implement block design with distinct contrasting colours. | Distinct block colours improve the user experience by easily distinguishing player moves. | Blocks must have visually distinct colours or patterns that are consistent throughout the game to avoid confusion. |
| 4 | Develop user-controller board rotation functionality. | Board rotation adds depth to the 3D gameplay by allowing planning from multiple viewpoints. | The board should rotate smoothly with intuitive controls, maintaining the player’s orientation and game focus. |
| 5 | Create interactive real-time block highlight system | Highlighting helps in decision making and enhances the game’s interactivity. | As the user navigates, the system should highlight the column clearly without obstructing the gameplay. |
| 6 | Design a win detection algorithm for connect 4 in 3D space. | A robust win detection algorithm is crucial for validating game outcomes. | The algorithm must quickly and accurately identify winning conditions across all planes and axes, and declare the winner without errors. |
| 7 | Establish a secure and efficient client server multiplayer network | A stable network is fundamental for multiplayer games, ensuring that players can compete against others without interruptions or connectivity issues. | The network must handle multiple games with minimal latency, ensuring game state synchronisation and user connectivity. |
| 8 | Develop a user authentication system. | Security measures are necessary to protect user data and ensure trust. | The system must use hash and salt for password to ensure user account is protected. |
| 9 | Design a user-friendly UI for game navigation and in-game action. | An intuitive is vital for a seamless user experience and game accessibility. | The UI should allow for easy access to all features, with clear visual cues and minimal response time for actions. |
| 10 | Construct a normalised database for storing game and user data | Data integrity and normalisation are key for reliable analytics and game functionality. | The database should enable efficient data retrieval and update and support complex queries for gameplay analysis. |
| 11 | Implement SQL operations for game data retrieval and user registration. | SQL operations are necessary for dynamic data interaction and game continuity. | Users should be able to register and retrieve their game data efficiently using simple SQL queries, ensuring data consistency. |
| 12 | Enable username uniqueness check during registration. | Unique usernames prevent user conflicts and support individual identities in-game. | The system must validate new usernames against the database to ensure they are already in use. |
| 13 | Design login verification to authenticate users. | Accurate login processes are essential for secure access to user accounts. | User credentials must be verified against stored encrypted data, with clear feedback provided for authentication success or failure. |
| 14 | Integrate an AI opponent with adaptive difficulty | An AI opponent allows for engaging gameplay when other players are not available or when players want to test their strategy | The AI should provide a challenge appropriate to the user’s skill level, using the Mini-Max algorithm to make strategic moves. |
| 15 | Create comprehensive guide for new players. | A guide helps new players understand the game, leading to a better user experience | The guide should be clear, concise, and cover all fundamental aspects of the game, including rules, controls, and strategies. |
| 16 | Create a replay menu which list the 10 most recent games the player has played, including detail about each match | A replay menu enhances the user experience by providing a history of games that can be reviewed, fostering better learning and strategy development. | The replay menu should display the 10 most recent games with detailed such as opponent name, outcome and moved played. |
| 17 | Develop a system that allows users to replay a match from the replay menu | Watching previous matches can be educational and entertaining, allowing players to analyse their gameplay and learn from past moves. | The user should be able to select and view a full replay of any listed match. The system should ensure the replay accurately reflects the game state transitions that occurred during the match. |
| 18 | Implement functionality for users to control the replay of a match, allowing them to navigate moves backward and forward | Providing control over the replay allows for a more in-depth analysis of specific moves and game phases, enhancing the learning experience. | Users should have controls to step through each move of the game, move by move, both backward and forward. The system should display the game state for each move accurately without any lag. |

## Summary Objectives List

1. Base Game Development
   1. Coin and 3D Board Creation
      1. Users should see and be able to interact with a 3D board.
      2. Users should be able to look at the board from different viewpoint.
      3. Users should easily identify which column they are hovering over.
   2. Win/ Validating Checking System
      1. The system should be able to identify when there is a connect 4, no connect 4 should be missed.
      2. The system should not allow more than 4 blocks to place within each column.
2. Client-Server Network Integration
   1. Users should be able to play online against other users.
   2. Users should be automatically assigned to a room with users that is +/- 100 of their score.
   3. Game room state should be tracked.
   4. Game room should not user to place a move when it is not their turn.
3. User Login and Authentication
   1. Implement a system to validate users with the correct username and password.
   2. Ensure password security through encryption, hashing, and salting.
4. User Interface (UI) Design
   1. Account Management UI
      1. New user must be able to open new user account.
      2. Old user must be able to log back into their old account.
         1. Account must store games that the user has played.
   2. Main Menu UI
      1. Users should be able to see their current score and their own username.
   3. Gameplay UI
      1. User should be able to see who they are playing again.
      2. User should be able to see what their opponent placed.
      3. User should know whose go it is.
   4. Endgame UI
      1. User should know who have won the match.
      2. Game should be saved into the database after it finishes.
      3. Users should have options to rematch or exit the game.
   5. Replay UI
      1. Display the 10 most recent games the user has played.
      2. Allow them to select which one they want to replay.
      3. User should be able to know how to move forward and backward in game state.
5. Database Management
   1. Database must be well structured to ensure efficient querying.
   2. Database must accurately save played games.
6. API Development
   1. Client should be able to do POST request handling.
7. Artificial Intelligence (AI) Implementation
   1. Users should be able to play the AI without any problem.
   2. Users should be able to vary the difficulty of the AI.

# Design

## IPOS Chart

This chart outlines what happens to the data in the new system at the most basic level, in terms of input/ output, processing, and storage.

|  |  |
| --- | --- |
| Input   * Player movements and selections via mouse/keyboard or touch input * User account information for login and registration * Option of Single Player or Multiplayer * Choice on the difficulty of the AI | Processing   * Game Logic to determine valid moves, check for winning conditions, and mange turn-taking. * Matchmaking algorithms to connect players for online gameplay. * Camera rotation logic * AI behaviour processing for single-player mode * Password processing – creating salt and hash to increase the security of the database. * Retrieve and display statistics from database. * Save games into database. * Networking logic (Client-Server logic) * User authentication and session management to ensure secure and persistent user connections. |
| Storage   * Player profiles, including game history and statistics. * Logs and records for gameplay analysis troubleshooting * Leaderboard rankings and scores | Outputs   * Visual Display of the 3D Connect 4 board and game pieces. * Visual Display of the position that the mouse is on * Game results and update to player statistics/ leader boards. * Display the winning connect 4. * Notification and alerts for matchmaking and turn notification. * Rematch option * Exit option. * Replay or review option * List all the moves that was play in Replay mode. |

## 3D Connect 4 - General System Flowchart

The following chart serves as a conceptual framework for the 3D Connect 4 game, adapted from the efficient model employed by Lichess. It is tailored to show the operational flow in the most abstracted form to our game’s architecture.

A notable difference in my game system is that user will be restricted from multiplayer gameplay unless they had login or registered, enhancing security and personalised experience – the reason for this is because this allows me to identify if a person is cheating as they will be referenced to their username.

While this general structure flowchart may bear resemblance to the Lichess system, the underlying details of my system, particularly the move verification logic (which will be expanded on under Game Duration process), are distinct due to the additional dimension in comparison to Lichess. Additionally, the Network logic will deviate considerably. Given the project’s scope, I will develop a robust local area network platform, contrasting Lichess’s global network reach. This approach is designed to accommodate our current resource allocation and time constraints, while still offering a comprehensive and engaging multiplayer experience.

**(Note: All the Golden coloured Processes will be further expanded upon, this is only an overview of the general system)**

The following flowcharts will demonstrate how I intended to design the processes that are coloured Golden in the General System Flowchart above. It depicts in the greatest depth how each of the processes will operate.

(Note: At this early stage, the client and server specifics are set aside for later discussion, ensuring a clear understanding the system’s core before integrating it into the various network component)

A diagram of a flowchart

Description automatically generated

## Registration System

The registration process is split into 3 main processes:

**Validate input:**

Before proceeding with the user registration, the input details must be validated. This validation includes checking if the username is unique and that the username and password meets the length requirement. If validation fails, the system flags an error and provides an error message to the user to correct their inputs.

**Encrypt password:**

Once the user details pass validation, the password encryption process begins. The system generates a unique salt – a random string of characters that adds additional security to the password hashing process, The password provided by the user is then hashed along with the

**Store Data:**

After the password is securely hashed with the salt, the user’s details, including the salt and the hash of the password, are stored in the system’s database. Storing the hash and salt ensures that even if the data is comprised, the actual password cannot be easily retrieved and used.

Flowchart:

Below is a flowchart for the registration process.

A diagram of a computer program

Description automatically generated

Hierarchy Chart:

This chart below shows the subroutines that are involved within the Register process.

A diagram of a computer system

Description automatically generated

## Login System

The process for login follows very similar principle as registration and it is done by 2 main parts:

Encrypt password:

Before proceeding with the verification, it must first be processed as the only the hash and salt value of the password is stored within the database. Unlike the registration process, a salt is not be created at this stage; instead, the existing salt stored in the database (associated with the user’s account from when they registered) is retrieved. The user’s password is then hashing with this salt. The resulting hash will be used in to compare against the stored hash in the database

Verify Inputs:

The system now validates the input credentials:

The username the user entered is checked to ensure an account with that username exists. The hash generated from the entered password and the retrieved salt is compared with the hash stored in the database during the registration process. If they match, it confirms that the user has entered the correct password.

Successful or Unsuccessful Login

If both the username exists and the hashed passwords match, the user is successfully logged into the system.

If there is a mismatch in the username or hashed passwords, the login attempt is denied, and the user will receive an error message indicating an incorrect username or password.

Flowchart:

A diagram of a software system

Description automatically generated

Hierarchy Chart:

A diagram of a computer

Description automatically generated

## Game Duration

The general flow of the game is very similar to all other turn base game that already exist in the market. Before going into depth and explaining the different component that is involved in managing the processes involving the managing the Game Logic. I want to show a general overview of what goes through within a game. As you can see below, the main feature of a turn base game, like my 3D Connect 4, is the Gameplay Loop which is involved this loop happens endlessly until there is a connect 4 achieved or when the board is full. Note that the golden coloured area will be further expanded later on in design.

A screenshot of a computer flowchart

Description automatically generated

UML Diagram

To gain a better understand of my design for the game logic I have created a UML diagram to show the different components interact with each other to create a smooth game experience for the users. It is important to note that this UML diagram shows the foundation of the Game Logic. Later on, when explaining the Multiplayer network solution, some of the game logic will be also adapted into the server class as the server will be handling validation instead of having the client to validate the move.

A screenshot of a computer

Description automatically generated

I have split the game logic into three main classes to deal with Game logic:

|  |  |
| --- | --- |
| Class | Description |
| Column Input | Its functionality is to detect player interactions with the game board. |
| Game Manager | This is the central hub; it orchestrates the various actions that take place throughout a match. It responds to user interactions, governs the turn-based mechanics, checks for win conditions, and updates the game state according with the help of the Playfield class. |
| Playfield | This script represents that game board’s data structure and logic. It keeps track of the positions of the blocks placed by each player. |

Below is the Process that goes through every time a game is initialise:

**Initialisation of Rows and Columns:**

A screenshot of a computer

Description automatically generated

The Game is predefined with a set number of rows and columns, which correspond to the x and z dimensions of the 3D grid. These dimensions are fixed and known at the start of the game, setting the boundaries within which players can drop their blocks.

**Determining the Layer with ValidMove method:**

You might have wondered how the programme knows which layer to place the block into. When a player attempts to place a block in a column, the crucial piece of information that needs to be calculated is the layer (y dimension) at which the block will come to rest. This is into predefined because it depends on how many blocks have already been placed in that particular column. The ValidMove method in the Playfield script takes the row and column as inputs and iterates from the top layer downwards to find the first empty space. This simulates the real-world gravity that a Connect 4 game would have, ensuring that blocks stack on top of each other.

**User Interaction with Column Input:**

As the user hovers the mouse over the visual representation of a column and clicks, the ColumnInput script, which is attached to each column, detects this interaction. It has the row and column values already associated with it. When clicked, it calls the ColumnPressed method in the GameManager Class – passing through the row and column value along.

**Handling the Move:**

The ColumnPressed method receives the row and column and must now determine the layer. It calls ValidMove on the Playfield instance, which returns the correct layer. With all 3 coordinates (row, column, layer) now known, the GameManager can proceed to the actual game action – dropping the physical block onto the Player’s Screen

**Executing the Block Placement with PlayBlock:**

Using the obtained layer information, the GameManager calls PlayBlock. This method will animate the dropping of the block to the calculated position and then depending on the currentPlayer Value, it drops the associated colour.

**Updating the Game State with DropBlock:**

Once the block has visually reached its destination, the Playfield DropBlock method is called, which updates the internal 3D array with block’s placement, effectively locking it in as part of the game state.

**Checking for a Win with WinCheck:**

Following the placement of the block, the Playfield triggers a check for a connect 4 on the board. This method performs a search through the array to detect four of a kind.

**Resolving the Turn with WinCondition and SwitchPlayer:**

If WinCheck finds a winning alignment, it informs GameManager’s WinCondition, which then proceeds to handle the end of the game, display the appropriate game over window and setting up for a rematch or exit.

If no win is detected, the game is still in progress, and it’s time to switch to the next player. This is done through SwitchPlayer method, which toggles the active player and updates the UI to reflect whose turn it is.

## Replay Logic

The replay system allows users to view a past game’s moves sequentially, enhancing their understanding. The replay feature is designed to fetch a historical game’s move sequence from a database and allow the user to navigate through these moves, both forwards and backwards.

Description:

You can think about the replay system to be split into three key steps.

Initialisation:

1. Start: The process begins when the user chooses to replay a specific game.
2. Initialise game Board: The game board is set up in the initial state with no pieces from either player, ready for the replay to begin.
3. Initialise Move History list: an API request is sent to the database to retrieve the moves made during a game. This list includes the coordinates for each move.
4. Display Game Details on Screen: Players’ name and their scores and who won the game are displayed on the screen to provide context to the replay being viewed.

Playback Control:

1. End or Continue: The system is consistently checking the weather player have clicked to exit button. If clicked, then the system returns the user back to the main menu.
2. Listen for Play Press: The system listens for user input to control the replay. There are two main inputs considered:
   1. Forward Press (right arrow on keyboard or right arrow button on screen): Advances the replay to the next move.
   2. Backward Press (left arrow on keyboard or right arrow button on screen): rewinds the replay to the previous move. Removing the move that was just played from the screen.

Move Execution:

1. Current move Number: The system keep track of the current move index number in the number sequence.
2. Find the move from the moves list: Based on the user input, the system finds the move from the history list. If the user pressed forward, it finds the next move; if backwards, the previous one.
3. Play Block on Screen: Once the move is identified, the corresponding block is visually played on the game board. If the move is a forward step, the block will be added to the board, and if backward, the last block added will be removed.

Increment/ Decrement Move Number:

* If the user is advancing through the replay, the move number is incremented to progress to the next move in the sequence.
* If the user is going back in the replay, the move number is decremented to return to the previous move.

**Flowchart**

Here is the flowchart of the system to give a better understanding of how the system run.

A diagram of a process

Description automatically generated

**Hierarchy Diagram**

The Hierarchy Diagram shows more in depth the different components that is involved within the Replay system.

A diagram of a computer

Description automatically generated

|  |  |
| --- | --- |
| Methods | Description |
| Listener | The listener is the update loop that will always listen to user input. When something is detected. It identifies whether it is to move forward or backwards, depending on which one it is, the Listener then call the according method. |
| OnFowardArrow | Increments the current move index by one. |
| OnBackwardArrow | Decrements the move current index by one |
| PlayBlock | Use to drop the physical block on the screen. |
| RemoveBlock | Use to remove the physical block on the screen. |
| GetMove | Use to retrieve the moves from the database. This will be in a form of an API request by the client to the database – this will be further discussed later. The data that returns is in a form of a Json string. |
| DeserialiseJson | Use to deserialise the Json string that was retrieved. And put all the move in a list. – this will be discussed further when talked about database |

## Outline Selection

As I delved into the design intricacies of my 3D Connect 4 game, it became apparent that guiding players through the complex three – dimensional environment was paramount. With this in mind, I conceptualised an ‘Outline Selection’ system that would assist in making the gameplay both intuitive and engaging. As a result, I designed OutlineSelection class to fulfil this need.

Justification:

* Navigation Clarity: Players need clear navigation cues in a 3D environment to interact effectively with the game elements. Without such cues, selecting the correct game pieces can be hard.
* User Expectation: Modern gaming interfaces often provide interactive feedback. Implementing such a feature meets the contemporary standards that users expect.

Design and Application:

* The ‘OutlineSelection’ class will use a raycasting mechanism to detect and highlight game pieces that the user’s cursor points to.
* It is designed to react when the cursor is over an object tagged as “Selectable”, and it will visually distinguish these objects by rendering an outline around them.
* Upon user selection, typically with a mouse click, the outline will persist on the selected piece to indicate an active state of interaction.

Raycasting:

Raycasting is an integral part of the OutlineSelection system. As a result, I believe that it is important to delve deeper into the concept of raycasting in Unity and it use within the OutlineSelection class for my 3D Connect 4 game.

What is Raycasting:

Raycasting is a function provided by Unity's physics engine that simulates the firing of an invisible ray from a point in space along a specified direction. The fundamental purpose of a raycast is to detect objects that are hit by this ray. It is a computational method often used in games and simulations to determine line-of-sight interactions, collision detection, or object selection.

Raycasting in OutlineSelection system:

In the context of OutlineSelection class for your game, raycasting is used to determine which game piece a playeris intending to interact with. Here’s how the class would employ raycasting:

* When the player moves the mouse, a raycast is triggered from the camera’s position towards the mouse position in the 3D game world.
* If the ray intersects with a game piece that has a collider and is tagged as selectable this game piece is considered hit by the ray.
* The class then uses this information to activate an outline around the game piece, providing visual feedback that the piece is interactable.
* If the player clicks the mouse button while a piece is highlighted, the class treats this as selection and maintain the outline to indicate that the piece is actively selected.

Flowchart:

A diagram of a flowchart

Description automatically generated

## AI – Minimax Algorithm

Below I am going to discuss about the Minimax Algorithm that I am going to be using for my AI bot within my game allowing user to play by themselves.

Description of Minimax Algorithm:

The Minimax algorithm is applied to two-player turn-based games, where the goal is to find the optimal move. An optimal move is about ensuring the best outcome for yourself even in the worst-case scenario: being when you opponent plays optimally as well.

It operates on a simple principle: maximize the possible score for one player while minimising that of the opponent. To do this, the algorithm uses a depth-first approach, the algorithm recursively generates a game tree, representing all potential future moves up to a certain depth.

Each move is simulated through generating a copy of the current state of the board and performing a move on it. It alternates between the roles of maximiser and minimiser, assessing the moves from each player's perspective. This will be later demonstrated below.

The way to assess the move is via evaluation function is key in this algorithm, assigning scores to the game states based on the assigned value. This part will be determined by the creator. When it encounters a terminal state (win state) or reaches the maximum depth, the algorithm backtracks, assigning values to parent nodes based on the best scores available to the maximiser or the lowest scores for the minimiser. After evaluating all options, the algorithm selects the move that offers the maximiser the highest score or the if it is a minimiser the lowest score.

Example:

In this example, it will explain the way in which the algorithm finds the best move. In every game, there are two players, and each one will be defined as Maximiser and Minimiser.

The Maximiser will try to get the Maximum possible score, and Minimiser will try to get the minimum possible score.

In this algorithm, it will apply Depth First Search – so imagine a game-tree, and we will go all the way through the leaves to reach the terminal node (the node which represent the end of the game). At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.

Step 1:

In the initial phase of the process, the algorithm constructs a comprehensive game tree, evaluating each of the end states using a pre-determined evaluation function to ascertain their values. Imagine the root of this tree, labeled 'A' and positioned at depth zero, as the starting point. If the player aiming for the highest score (maximizer) goes first, we start with the assumption that the maximizer's initial value is at its lowest possible, which is -∞. Conversely, the player aiming to minimize the score (minimizer) follows, starting with an initial value at its highest possible, which is +∞.Diagram of a diagram of a terminal values

Description automatically generated

Step 2:

Initially, the evaluation value for the player seeking the highest score, known as the Maximiser, is set at -∞. As we progress through the terminal states, we compare the values at each terminal state with the Maximiser's initial value. This comparison is carried out to ascertain the optimal values at the higher nodes, effectively searching for the greatest value among all possible outcomes in the game tree.

* For node D         max(-1, -∞) => max(-1,4)= 4
* For Node E         max(2, -∞) => max(2, 6)= 6
* For Node F         max(-3, -∞) => max(-3,-5) = -3
* For node G         max(0, -∞) = max(0, 7) = 7

Diagram of a diagram of a diagram

Description automatically generated

Step 3:  
Next, the process shifts to the Minimiser. In this stage, the Minimiser compares the values of all potential nodes to +∞. The purpose of this is to establish the values for the nodes in the third layer, effectively determining the least possible outcomes from the available nodes to proceed in the game tree.

* For node B = min(4,6) = 4
* For node C = min (-3, 7) = -3

A diagram of a diagram

Description automatically generated

**Step 4**

Now it's a turn for Maximiser, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

* For node A max(4, -3) = 4

As you can see here, even though node I does not have the maximum terminal value, however, since the algorithm is assuming optimal play by the user, it will consider the worst case, for the next move, thus minimising the amount.

Diagram of a diagram of a diagram

Description automatically generated

Properties of Mini-Max algorithm:

* **Complete-** Min-Max algorithm is Complete. It will find a solution (if exist), in the finite search tree.
* **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **O(bm)**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
* **Space Complexity-** Space complexity of Mini-max algorithm is also like DFS which is **O(bm)**

**Evaluation method:**

I have mentioned above how important the evaluation function is to make a good minimax algorithm. A good evaluation is what defines whether an algorithm is good or bad algorithm.

In the example above the evaluation values were just assigned by random to show how the algorithm functions. In the actual implementation of the algorithm for 3D connect 4, the evaluation method will be involving 3 main parts:

* 2 streak : +/- 50 points
* 3 streak : +/- 100 points
* 4 streak : +/- 1000 points

As you can see depending on the streak, a value is given to it. The evaluation method will iterate through the terminal state board to find whether any of these streaks are discover during the iteration and return the value once finished.

The evaluation method will not only involve one loop, instead it will be multiple checking through different diagonals, horizontals and verticals through different dimensions, similar to the validation check. Thus, the evaluation method will be a placeholder where it sums up the score and return the total.

You might question now: Why the evaluation method does not simply assign a single value, such as +/- 1, given that the algorithm’s objective is to find a winning state.

Let take an example and consider the game’s opening. Although, there is only a finite of moves to make, the deviation of the game from there is near infinite – the deviation becomes exponentially large as it goes deeper into the future of the game. Hence, this is where depth plays a critical role and why I have implemented more than one score value. By adding the maximum depth (maxDepth/ depth), it determines how many moves ahead the algorithm is evaluating - I.e. a depth of two will be applying two moves and then evaluating the state of that board. This depth remove the chance of the recursive function entering a never ending loop, offering the algorithm a place to exit. It gives the algorithm the balance between time and accuracy.

**Pseudocode:**

Below I have written in pseudocode for the main algorithmic part within the minimax script that is going to be created.

|  |
| --- |
| FUNCTION minimax(node, depth, maxmimisingPlayer)  IF depth == 0 or node is a terminal node THEN  Return static evaluation of node  IF MaximisingPlayer THEN  maxEva ← -infinity  FOR child FROM node  eva ← minimax(child, depth-1, false)  maxEva = max(maxEva, eva)  RETURN maxEva  ELSE  minEva ← +infinity  FOR child FROM node  Eva ← minimax(child, depth-1, true)  RETURN minEva  ENDFUNCTION |

#### Alpha Beta Pruning

After discussing the bases of the Minimax algorithm, I am going to explain an optimisation technique for the minimax algorithm. It is use in order to reduce the computation time, therefore allowing the algorithm to search much faster and even go into deeper levels in the game tree.

Alpha Beta Pruning is used to cut of branches in the game tree which need not be searched because already exits a better move available. In this technique 2 extra parameters in the minimax function, called alpha and beta.

Alpha: the best value that the maximiser can guarantee at that level or above.

Beta: the value that the minimiser currently can guarantee at that level or below.

Pseudocode:

|  |
| --- |
| FUNCTION minimax(node, depth, isMaximizingPlayer, alpha, beta):  IF node is a leaf node :  return value of the node    IF isMaximizingPlayer :  bestVal ← -INFINITY  FORchild FROM node :  value ← minimax(node, depth+1, false, alpha, beta)  bestVal ← max( bestVal, value)  alpha ← max( alpha, bestVal)  IF beta <= alpha THEN  break  RETURN bestVal  ELSE:  bestVal ← +INFINITY  FORchild FROM node :  value ← minimax(node, depth+1, true, alpha, beta)  bestVal ← min( bestVal, value)  beta = min( beta, bestVal)  IF beta <= alpha THEN  break  RETURN bestVal  ENDFUNCTION |

Zobrist HashingAlongside of Alpha Beta Pruning, Zobrist hashing is also used within the algorithm to reduce computation time of the algorithm. Zobrist Hashing is a hashing algorithm used in transposition table. This table store the evaluated values of previous board states, so that if they are encountered again, it can simply retrieve the stored value from the hash table.

How it works:

**Step 1** –Initialise a Zobrist Table Setup:

This table is used to store unique long values for each possible state of each cell in the game board. The dimensions of the Table will be corresponded to the game that is being implemented in. In our case our dimensions have 4 layers, rows, columns and two players. This initialise therefore then allow us to generate a unique hash code for a given board configuration.

**Step 2 –** Generating Random Values:

For each cell and player combination, the algorithm will generate a random long value. This value is generated by filling a byte array with random bytes and then converting them into a long integer – this then is the value that represent a board state.

**Step 3** – Computing Hash Codes:

Whenever the algorithm needs to evaluate a board state, it will need to compute its hash code. This is done by iterating over every cell in the board, and then checks if a piece occupies the cell. It then runs an XOR gate with the corresponding value from the Zobrist table with an accumulating hash value – this results in a generation of a unique hash value representing the current board state.

**Step 4** – Transposition table:

At each recursion of the Minimax function, the function will first check the transposition table – dictionary mapping hash codes to evaluation scores – to see if the current board’s hash code already has a computed evaluation. If it does, it just immediately returns this value, skipping the need to evaluate the board state. Else, the algorithm proceeds with Minimax evaluation.

Pseudocode:

|  |
| --- |
| # Constants for pieces  Empty ← 0  Player1\_Piece ← 1  Player2\_Piece ← 2  # Initialize Zobrist table for a 4x4x4 board  SUBROUTINE init\_zobrist():  table ← a 3-d array of size 4x4x4 for board positions, and an additional dimension FOR pieces (3 types including empty)  FOR layer from 1 to 4:  FOR row FROM 1 to 4:  FOR col FROM 1 to 4:  FOR piece FROM 0 to 2:  table[layer][row][col][piece] ← random\_bitstring()  table.current\_player = random\_bitstring() # Additional state if needed for whose turn  ENDSUBROUTINE  # Compute hash for a given 3D board state  FUNCTION hash(board, is\_player1\_turn):  hash ← 0  IF not is\_player1\_turn THEN:  hash ← XOR table.current\_player  FOR layer FROM 1 to 4:  FOR row FROM 1 to 4:  FOR col FROM 1 to 4:  IF board[layer][row][col] ≠ Empty THEN:  piece ← the piece at board[layer][row][col]  hash ← h XOR table[layer][row][col][piece]  RETURN hash  ENDFUNCTION |

## Network

### General Network Design

The networking system behind the game enables multiplayer functionally, allowing players to compete in real-time. This system is structured around a client-server model, which characterised by a central server that manages the game state, paired with multiple client each present a user in the game.

In this architecture, the server handles the majority of critical game function and logic, ensuring that the game state is consistent across all clients. In both client and server, each have their according associated Network Handler which is used to compute any data and complete any task that is needed to be done. Furthermore, there is a class called NetUtility which acts as a central hub for all communication messages.

Below is a visual representation of the Network architecture:  
Note: It is obvious that Client and Server here has their individual Network Handler, however, it is important to note that they both also have their individual NetUtility and NetMessage components. The reason, in the diagram, I have showed it so both Server and Client points towards the same NetUtility is to reduce redundancy in the diagram.

A diagram of a network

Description automatically generated

|  |  |
| --- | --- |
| Components | Description |
| NetworkManager | Deals with joining and hosting page, connecting the client to the server, and allowing the server to be created. |
| Server | The server performs the heavy lifting, such as processing game logic, maintaining the state of the playfield, and validating player actions. Its most important tasks is also to ensure that messages are broadcasted to the correct client’s. |
| Client | The client focuses on presenting the game state to the user, capturing user input, and sending these inputs to the server. |
| NetUtility | Acts as the central hub for all the Netmessages – making it easier for the client and server to interpret the messages. |
| NetEndGame/ NetKeepAlive/ NetMakeMove/ NetRematch/ NetStartGame, NetMessages: | These are the only messages that can be sent or received by the client and server. These messages are used to send various messages to the client or server so that they can act upon it. |
| ServerNetworkHandler/ ClientNetworkHandler | This is where it interprets incoming messages, execute appropriate response, ad handle the transmission of game data. |

### Client Side

In this project, the client should be able to do several things:

* Establishing a connection to the server.
* Disconnecting from the server when necessary.
* Handling incoming messages from the server.
* Sending request messages to the server.

These are the 4 main core features that the clients be able to do. To do these tasks, I have divided the responsibilities into two classes.

The first class (called “Client”) is responsible for managing the connecting state and facilitating message transmission. The second (called “ClientNetworkHandler”) specialise in unpacking the different messages that the server may send to the client and processing them accordingly.

This separation allows for clear and focused logic implementation within each class, particularly in the case of the ClientNetworkHandler, where message handling logic is required due to the variety of possible server messages. The kind of messages will be further discussed later when I discuss about NetMessages.

Alongside these core features, the client also performs another two vital tasks:

1. periodically checking the status of its connection to the server.
2. Holding their username, userid, score and the activity they are using (i.e. AI mode, multiplayer)

Status Checking

If the connection is found to be inactive, the client initiates a graceful disconnection. Additionally, to ensure smooth communication, the client regularly communicates its active status to the server.

Holding User information

To do this, I have designed a class (called “DBManager”), this is used to store the different information about client’s activity.

#### Client Class

Here are the different methods that is going to be implemented for the client class to be able to connect, disconnect to and from the server, and to send message to the server.

|  |  |
| --- | --- |
| Method | Description |
| Init | Initialises the network driver and attempts to connect to the server at the specified IP address and port. It sets up the client as active and registers to network event. |
| Shutdown | Cleans up the network driver and other resources when the client is shutting down, ensuring a clean disconnect from the server. It also unregisters from network events to present any potential memory leaks or callbacks to destroyed object. |
| Update | The main update loop that checks for the active state of the client, updates the network driver, checks the connection status, and processes any incoming network events. |
| CheckAlive | Checks if the connection to the server is still alive. If not, it triggers the connectionDropped event and shuts down the client to clean up resources. |
| UpdateMessagePump | Processes all incoming network events, handling connect, data, and disconnect events. On connect, it sends a welcome message to the server. On data, it handles incoming messages, and on disconnect, it cleans up the client. |
| SendToServer | Serializes and sends a message to the server, encapsulating the data in a format that the network driver can transmit. |
| OnKeepalive | Responds to keep-alive messages from the server by echoing them back, indicating that the client is still active and connected. |

**Pseudocode:**

|  |
| --- |
| CLASS Client  isActive ← FALSE  SUBROUTINE Init(ip: STRING, port: INTEGER)  driver ← CREATE NetworkDriver  endpoint ← PARSE the IP and the port number to create a network endpoint  connection ← Attempt to Connect to server at endpoint using driver  OUTPUT "Attempting to connect to Server on " + endpoint.Address  isActive ← TRUE  CALL RegisterToEvent  ENDSUBROUTINE  SUBROUTINE Shutdown()  IF isActive THEN  Dispose Driver  isActive ← FALSE  connection ← RESET NetworkConnection To Default State  ENDIF  ENDSUBROUTINE  SUBROUTINE Update()  Perform and Complete Network Update using driver  CALL CheckAlive  CALL UpdateMessagePump  ENDSUBROUTINE  SUBROUTINE CheckAlive()  IF Client and Server connection NOT Created AND isActive THEN  OUTPUT "Lost connection to server"  CALL Shutdown  ENDIF  ENDSUBROUTINE  SUBROUTINE UpdateMessagePump()  stream ← DataStreamReader #read data from the stream  cmd ← TypeOfNetworkEvent  WHILE (cmd ← (attempt to RETRIEVE data from the current connection)) AND (cmd IS NOT Empty)  IF cmd IS Connect THEN  CALL SendToServer(username, score) #userInfo sent  OUTPUT "We are connected"  ELSEIF cmd = NetworkEvent.Type.Data THEN  CALL NetUtility's OnData(stream, Default Connection state)  ELSEIF cmd IS Disconnect THEN  OUTPUT "Client disconnected from server"  CALL Shutdown  ENDIF  ENDWHILE  ENDSUBROUTINE  SUBROUTINE SendToServer(msg: NetMessage)  DECLARE writer AS DataStreamWriter    START preparing data to send TO connection, USING writer  SERILISE the message into write  SEND data USING writer  ENDSUBROUTINE  SUBROUTINE OnKeepAlive(nm: NetMessage)  SendToServer(echo message to indicate they are alive)  ENDSUBROUTINE  ENDCLASS |

#### ClientNetworkHandler

Here is a list of the different methos that perform different actions according to the message that is received.

|  |  |
| --- | --- |
| OnWelcomGameClient | Deal with NetWelcome Message:  It sets up initial game state based on server assignments, such as player teams. |
| OnStartGameClient | Deal with NetStartGame Message:  Triggers the game’s commencement from a client perspective, handling the UI transitions and game setup (i.e. displaying Black and White player’s name and their constituent scores). |
| OnMakeMoveClient | Deal with NetMakeMove Message:  Update the game state based on moves made by the opponent, communicated by the server. This ensures that the view of the game for both client within the room is consistent with server’s authoritative state. |
| OnRematchClient | Deal with NetRematch Message:  Manages the logic for players requesting for rematch, updating local states to reflect what the opponent do after the game finish.  For example:  If they want rematch, it will display that the player want rematch on the UI |
| OnEndGame | Deal with NetEndGame Message:  It indicates to the clients that the game within the room has ended, and the necessary UI is then Display |

|  |
| --- |
| CLASS ClientNetworkHandler  currentTeam ← 0  playerRematch ← [FALSE, FALSE]  SUBROUTINE OnWelcomeClient(NetMessage)  #Recieve the connection message  nw ← Unpack NetMessage [NetMessage will be NetWelcome]  #assignTeam  currentTeam ← nw.AssignedTeam [Type INT]  ENDSUBROUTINE    SUBROUTINE OnStartGameClient(NetMessage  GoToOnlineGame    nsg ← UNPACK NetMessage [NetMessage will be NetStartGame]    Set Up User display for the received pack  ENDSUBROUTINE  SUBROUTINE OnMakeMoveClient(NetMessage)  mm ← UNPACK NetMessage [NetMessage will be NetMakeMove]  PlayBlock(mm.lastMoveRow, mm.lastMoveColumn, mm.lastMoveLayer, mm.teamId)  ENDSUBROUTINE  SUBROUTINE OnRematchClient(NetMessage)  rm ← UNPACK NetMessage [NetMessage will be NetRematch]  playerRematch[rm.teamId - 1] ← rm.wantRematch == 1; #Type Bool if == 1 -> true  IF (rm.teamId != currentTeam) THEN  DISPLAY PlayerWantRematch On Screen  ENDIF    IF (playerRematch[0] && playerRematch[1]) THEN  #Physical ResetBoard  GameReset();    #Change who goes first  IF currentTeam == 1 THEN  currentTeam ← 2  ELSE  currentTeam ← 1  ENDIF  ENDSUBROUTINE  SUBROUTINE OnEndGameClient(NetMessage)  em ← UNPACK NetMessage [NetMessage will be NetEndGame]  IF (em != null AND em.EndGame == 1) THEN  DISPLAY EndGameMenu  ENDSUBROUTINE  ENDCLASS |

#### Storing Client’s Information - DBManager

Effective management of client data is essential for maintain seamless user experience is highly important. To address this, I designed DBManager to manage user information effectively post-login. This acts as a central hub that maintains persistent storage of critical data such as user identification and gameplay statistics, which are necessary for continuity and personalisation. Moreover, specific Boolean flags within the DBManager are utilise to keep a real-time track of client’s status, allowing the system to adapt dynamically to the user’s current mode of interaction.

**Advantage and Design of DBManager**

1. Centralised User Data Management

I know that I have not yet discussed database management very much yet. However, after designing the client side of the network model, I realised that I still have not yet have an area that can store the user’s data and activity.

Thus, as a result, I decided to implement the DBManager class. This acts as a central repository for the client’s user-related data – including username, userID, score and flags that represent the activity the user is doing. By doing this, I ensure that any part of my application can access and modify user data in a consistent manner, thereby reducing any potential manner.

1. Static Class design

Utilising a static class design for the DBManager ensures that the user data remains persistent throughout the lifetime of the application’s session. Since static classes in C# are initialised once and remain in memory, they are naturally suited for maintaining a consistent state. This is particularly important in networked applications where use states need to be tracked and managed across different parts of the application.

1. Efficiency and Performance

The DBManager class improves the efficiency of the application by minimising the need for repetitive data retrieval operations from the server. Since it holds crucial user data, it serve data requrests from various components of the application without the overhead of repeated database queries, thereby reducing network traffic and server load.

Fields within DBManager:

|  |  |  |
| --- | --- | --- |
| Fields | Data Type | Description for Tracking/ Storing |
| username | Static string | Identifies the user; necessary for personalisation and user identification across session + online mode (sent across the server) |
| score | Static integer | Records the user’s score; user for game progress – especially used on multiplayer online mode + online mode (sent across the server) |
| userid | Static int | Unique identifier for the user; used for server database operations and to link data to specific users + online mode (across the server) |
| online | Static bool | Indicates if the user is currently online; can be used to track active users and manage live features. |
| ai | Static bool | Flags whether the user is playing against AI; helps in adjusting game behaviour and difficulty. |
| difficulty | Static int | Stores the chosen difficulty level for the AI; used to configure AI complexity and user experience. |
| LoggedIn | void | A property that checks if a user is logged in based on the presence of username; critical for access control and session management. |

|  |
| --- |
| STATIC CLASS DBManager  #Static public variables  username ← None  score ← 0  userid ← 0  online ← FALSE  ai ← FALSE  difficulty ← 0    FUNCTION is\_logged\_in()  #Check if the user is logged in based on the presence of a username  RETURN DBManager.username is not None  END FUNCTION  FUNCTION LogOut()  #log out the current user by resetting the username to None  Username ← None  END FUNCTION  END STATIC CLASS |

### Server

The server component operates similarly to the client, but it undertakes additional responsibilities. Similar to the client, the server component divided into two classes: The server class – responsible for tasks with connections, ServerNetworkHandler class – responsible for packaging different messages to be sent off.

The server undertakes additional responsibilities due to the nature of a thin client network:

* Connection Management: The server accepts incoming networking connections and regularly verifies active connections to eliminate any stale connections.
* Broadcasting Messages: The server is responsible of sending the correct message to the correct room or client.
* Packaging Messages: The server is responsible of packaging messages so that it can be sent through to the client.
* Room and Player Management: Assigning connecting players to appropriate rooms and maintaining a list of game rooms and user information. Each room keeps track of its players, while user data is stored within the server class.
* Game State management and Gameplay Logic: A nested class, called “GameRoom”, is utilised, responsible for representing individual game rooms and handling gme state and logic.
* Data Storage: Handling the storage of game moves and outcome in the backend.

Checking Connection (Connection Management):

As mentioned earlier in the client description, there's a vital aspect that the client is responsible for: signalling to the server that they are still active. This crucial function is where the server comes into play. It periodically dispatches a message to clients, usually at intervals like every 20 seconds. Upon receiving this message, clients promptly echo it back to the server. This action serves as a confirmation of their active status. Essentially, this reciprocal communication acts as a mechanism, allowing the server to effectively detect and manage stale connections.

**Nested Classes within the Server Class:**Within the server class, two nested classes, called ‘GameRoom’ and ‘UserInfo’, are created to store various data:

GameRoom:

As I have described above when describing the various tasks the server has, the GameRoom class involves all the game logic and validation checks within it. The reason for this nested class is because at each moment, there will always be a chance of having one or more room in parallel happening. As a result, this meant that it had to be a class object so that there can be multiple instance of board, thus allowing concurrent games.

UserInfo:

The UserInfo class is used to store the user information for all client’s that are connected to the server. Within the UserInfo class, it contains the userid, username and score of the user. These data are crucial within the server as they are used to assign players to room and to sent to opponent so that they know who they are playing against.

|  |  |
| --- | --- |
| Subroutines/ Functions/ Class | Descriptions |
| Init | Initialises the server, setting up network drivers |
| Shutdown | Cleans up resources, stopping the server |
| OnDestroy | Unity callback that ensures server, shutdown when the server object is destroyed |
| Update | Main loop handling keep – alive messages, connection maintenance, accepting new connections and processing incoming messages |
| KeepAlive | Sends keep-alive messages to all players in all rooms to ensure that all clients are still connected |
| CleanUpConnections | Removes non – active connections from the list |
| AcceptNewConnections | Accepts incoming connections |
| UpdateMessagePump | Processes incoming network messages, handling data and disconnect events |
| Broadcast | Sends a message to all clients in a specified list of player IDs |
| GameRoom | Nested Class which is within the Server Script:  Handle game logic for specific room:   * Maintaining a 3D game board state * Processing player moves * Checking win conditions * Managing player turns * Resetting the game board for new games |
| UserInfo | Nested Class which is within the Server Script:  A UserInfo class is used to store the user information. An object is created for every user connected. |

**Pseudocode:**

|  |
| --- |
| CLASS Server  rooms ← {INTEGER : LIST OF INTEGER}  gameRooms ← {INTEGER, CLASS : GameRoom}  usernames ← {INTEGER, STRING}  userInfo ← {INTEGER, CLASS : UserInfo}  roomCount ← 1  connections ← [] #Hold all the Connections  isActive ← FALSE  CONSTANT keepAliveTickRate AS FLOAT ← 20.0f # interval time  Class UserInfo(self, username, score, userid) #UserInfo Nested Class  Username ← self.username  score ← self.score  userid ← self.userid  SUBROUTINE Init(port: INTEGER)  driver ← CREATE NetworkDriver  endpoint ← anyIPv4 #determine who can connect  IF driver NOT Bind to endpoint THEN  OUTPUT "Unable to bind on port " + port  RETURN  ELSE  ListenToDriver()  OUTPUT "Listening on port " + port  ENDIF  isActive ← TRUE  ENDSUBROUTINE  SUBROUTINE Shutdown()  IF isActive THEN  Dispose Driver  Dispose connections on the server  isActive ← FALSE  ENDIF  ENDSUBROUTINE  SUBROUTINE OnDestroy()  CALL Shutdown  ENDSUBROUTINE  SUBROUTINE Update()  IF NOT isActive THEN  RETURN  ENDIF  CALL KeepAlive  CHECK for incoming network data  UPDATE for the network driver  CALL CleanupConnections  CALL AcceptNewConnections  CALL UpdateMessagePump  ENDSUBROUTINE  SUBROUTINE KeepAlive()  IF Time.Now - lastKeepAlive > keepAliveTickRate THEN  lastKeepAlive ← Time.Now  FOR room IN rooms  CALL Broadcast(new NetKeepAlive(), room.Value)  ENDFOR  ENDIF  ENDSUBROUTINE  SUBROUTINE CleanupConnections()  FOR i FROM 0 TO (LEN(connections) - 1)  IF NOT connections[i].IsCreated THEN  connections.RemoveAtSwapBack(i)  i ← i - 1  ENDIF  ENDFOR  ENDSUBROUTINE  SUBROUTINE AcceptNewConnections()  DECLARE c AS NetworkConnection  WHILE (c ← driver.Accept()) ≠ default(NetworkConnection)  connections.Add(c)  OUTPUT "Accepted a new connection"  ENDWHILE  ENDSUBROUTINE  SUBROUTINE UpdateMessagePump()  DECLARE stream AS DataStreamReader  FOR i FROM 0 TO (LEN(connections) - 1)  DECLARE cmd AS NetworkEvent.Type  WHILE (cmd ← (attempt to RETRIEVE data from the current connection)) AND (cmd IS NOT Empty)  IF cmd IS Data THEN  CALL OnData(stream, connections[i])  ELSEIF cmd IS Disconnect THEN  OUTPUT "Client disconnected from server"  # Handle disconnection  connections[i] ← default(NetworkConnection)  IF connectionDropped IS NOT NULL THEN  CALL connectionDropped REMOVE connection from the Dictionaries  ENDIF  ENDIF  ENDWHILE  ENDFOR  ENDSUBROUTINE  SUBROUTINE Broadcast(msg: NetMessage, playerIds: LIST OF INTEGER)  IF msg IS NULL THEN  OUTPUT "Trying to broadcast a null message"  RETURN  ENDIF  OUTPUT "Number of connections: " + LEN(connections)    FOR i FROM 0 TO (LEN(connections) - 1)  IF connections[i] IsCreated AND playerIds Contains(connections[i]) THEN  OUTPUT "Sending " + msg.Code + " to: " + connections[i].InternalId  CALL SendToClient(connections[i], msg)  ENDIF  ENDFOR  ENDSUBROUTINE  ENDCLASS |

**GameRoom:**

|  |
| --- |
| CLASS GameRoom  CONSTANT rows ← 4  CONSTANT columns ← 4  CONSTANT layers ←4  CONSTANT winlength ←4  coordinateList ← []  currentPlayer ← 1  connectionID ← []  usernames ← []  board = [  [  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ],  [  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ],  [  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ],  [  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0],  [0, 0, 0, 0]  ]  ]  SUBROUTINE GameRoom(id: LIST OF INTEGER)  connectionID ← id  # Other initialisation code...  ENDSUBROUTINE  FUNCTION WinCheck() RETURNS BOOLEAN  IF z\_axis\_Check() OR x\_axis\_Check() OR z\_x\_axis\_Check() OR  y\_axis\_Check() OR y\_z\_axis\_Check() OR x\_y\_axis\_Check() OR  x\_y\_z\_axis\_Check() THEN  RETURN TRUE  ELSE  RETURN FALSE  ENDIF  ENDFUNCTION  #Different checks written here    SUBROUTINE DropBlock(layer, row, column, player: INTEGER)  board[layer, row, column] ← player  coordinateList.ADD(NEW Vector3Int(row, column, layer))  IF WinCheck() THEN  # Perform game over actions  OUTPUT "WinCondition: true"  SAVE coordinateList into Database  ELSE  SwitchPlayer()  ENDIF  ENDSUBROUTINE  SUBROUTINE SwitchPlayer()  currentPlayer ← IF currentPlayer = 1 THEN 2 ELSE 1 ENDIF  OUTPUT "Switched to Player " + currentPlayer  ENDSUBROUTINE  ENDCLASS |

**Network Controller:**

**ServerNetworkHandler:**

|  |  |
| --- | --- |
| OnWelcomeServer | Handles client connections by assigning them to rooms and teams, based on the NetWelcome message. It updates user info on the server, assigns players to rooms, and broadcast the NetStartGame message to all players in a room once it is full, starting the game |
| OnMakeMoveServer | Processes move messages (NetMakeMove) from clients. If the move is valid, it updates the game state in the relevant GameRoom and broadcasts the move to the other player in the room |
| OnRematchServer | Handles rematch requests from clients. If both players in a room request a rematch, it resets the game board in the GameRoom and prepares for a new game. |

|  |
| --- |
| CLASS ServerNetworkHandler  playerRematch = (FALSE, FALSE)  SUBROUTINE OnWelcomeServer(NetMessage, NetworkConnection)  nw ← Unpack NetMessage [NetMessage will be NetWelcome]  score ← nw.score  username ← nw.username  AssignedTeam ← nw.AssignedTeam  CALL UpdateUserInfo(InternalId, username, score)  CALL AssignPlayerToRoom(NetworkConnection)  FOR room IN rooms  IF room.Value CONTAINS InternalId THEN  AssignedTeam ← playerIndex + 1  BREAK  ENDIF  ENDFOR    CALL SendToClient(NetworkConnection, NetMessage)  FOR room IN rooms  IF LEN(room.Value) ← 2 AND room.Value CONTAINS InternalId THEN  nsg ← NEW NetStartGame() [Reference to NetStartGame]  # Always Assuming the player in index 0 is Black and 1 is White  userInfoBlack ← userInfo[B\_Player\_InternalId]  userInfoWhite ← userInfo[W\_Player\_InternalId]    # Populate nsg with usernames and scores  nsg.scoreBlack ← userInfoBlack.Score  nsg.scoreWhite ← userInfoWhite.Score  nsg.usernameBlack ← userInfoBlack.Username  nsg.usernameWhite ← userInfoWhite.Username    CALL Broadcast(nsg, room.Value : List)  BREAK  ENDIF  ENDFOR  ENDSUBROUTINE  SUBROUTINE OnMakeMoveServer(NetMessage, NetworkConnection)  mm ← UNPACK NetMessage [NetMessage will be NetMakeMove]  FOR room IN rooms  IF LEN(room.Value) ← 2 AND room.Value CONTAINS InternalId THEN  Room = GetGameRoom(room.Key) [Reference to Class GameRoom in Server script]  IF mm.teamId ← Room.currentPlayer THEN  CALL Broadcast(msg, room.Value)  CALL Room.DropBlock(mm.lastMoveLayer, mm.lastMoveRow, mm.lastMoveColumn, mm.teamId)  ENDIF  BREAK  ENDIF  ENDFOR  ENDSUBROUTINE  SUBROUTINE OnRematchServer(NetMessage, NetworkConnection)  FOR room IN Server.Instance.rooms  IF LEN(room.Value) ← 2 AND room.Value CONTAINS InternalId THEN  rm = Unpack NetMessage [NetMessage will be NetRematch]  playerRematch[rm.teamId - 1] ← (rm.wantRematch == 1)  IF playerRematch[0] AND playerRematch[1] THEN  gameRoom ← GetGameRoom(room.Key) [Reference to Class GameRoom in Server script]  gameRoom.currentPlayer ← 1  CALL gameRoom.Reset\_Board()  ENDIF  BREAK  ENDIF  ENDFOR  ENDSUBROUTINE  ENDCLASS |

**NetMessages:**

NetMessages are the only messages that the client and the server can sent and receive. By employing NetMessage structure, I was able to establish the foundation for creating ClientNetworkHandler and ServerNetworkHandler, which are tasked with handling each message.

The structure of the NetMessages shares common attributes:

* Serialisation
* Deserialisation
* Received on Client
* Received on Server

These core functionalities form the backbone of every NetMessage and allow various data to be sent through the network seamlessly.

NetMessages:

Below there are the different type of NetMessages I have designed to be implemented within the Network.

|  |  |
| --- | --- |
| **NetMessages** | **Purpose** |
| NetWelcome | Typically used for initial greeting or handshakes between the client and server upon connection. It contains information like client’s username |
| NetKeepAlive | Used to maintain an active connection between the client and server, ensuring that both ends are still responsive and connected |
| NetRemtach | Sent when a player requests a rematch. This message can be used to reset the game state for another round without needing to reconnect |
| NetStartGame | Indicates the start of a new game. |
| NetMakeMove | Used when a player makes a move in the game. It contains the details of the move [row, column, layer, teamId] |
| NetMessage | A general or base class for network messages. This might be a parent class from which other specific message types inherit |
| NetEndGame | Sent when a winner is determined. This get sent to the client so that their rematch screen can be enabled. |

**Structure of NetMessage Pseudocode:**

|  |
| --- |
| CLASS NetMessage  PROPERTY Code AS OpCode  SUBROUTINE Serialise(writer: REFERENCE TO DataStreamWriter)  CALL writer.WriteByte(CAST Code TO BYTE)  ENDSUBROUTINE  SUBROUTINE Deserialise(reader: DataStreamReader)  # Implementation for deserialising data from the reader  ENDSUBROUTINE  SUBROUTINE ReceivedOnClient()  # Implementation for actions to take when message is received on client  ENDSUBROUTINE  SUBROUTINE ReceivedOnServer(cnn: NetworkConnection)  # Implementation for actions to take when message is received on server  ENDSUBROUTINE  ENDCLASS |

**NetUtility:**

In addition to NetMessages I added a NetUtility. It functions as a centralised location for all network related events. It defines a series of delegates or events for different types of network messages (i.e. player actions, game state updates, connection events). This centralisation makes it easier to manage and respond to these events throughout various parts of the game.

Different scripts can register with NetUtility to listen for specific network events. Alongside, it provides consistency within the network event and makes maintaining and updating the network code more manageable.

NetUtiltiy will be a static class as there is always only one single instance of this and there will not be any inheritance of this class.

Delegation in NetUtility:

Delegation was a key principle that is within NetUtility to receive and process the data. Instead of ClientNetworkHandler or ServerNetworkHandler processing the raw data directly, it delegates the tasks of serilisation and deserialization to the respective NetMessage. This division of responsibility allows for cleaner, more organised code.

**Pseudocode:**

|  |
| --- |
| ENUMERATION OpCode  KEEP\_ALIVE ← 1  WELCOME ← 2  START\_GAME ← 3  MAKE\_MOVE ← 4  END\_GAME ← 5  REMATCH ← 6  ENDENUMERATION  CLASS NetUtility  PROCEDURE OnData(stream: DataStreamReader, cnn: NetworkConnection, server: Server OPTIONAL)  DECLARE msg AS NetMessage  DECLARE opCode AS OpCode ← stream.ReadByte()  SWITCH opCode  CASE KEEP\_ALIVE  msg ← CREATE NEW NetKeepAlive WITH stream  CASE WELCOME  msg ← CREATE NEW NetWelcome WITH stream  CASE START\_GAME  msg ← CREATE NEW NetStartGame WITH stream  CASE MAKE\_MOVE  msg ← CREATE NEW NetMakeMove WITH stream  CASE END\_GAME  msg ← CREATE NEW NetEndGame WITH stream  CASE REMATCH  msg ← CREATE NEW NetRematch WITH stream  DEFAULT  OUTPUT ERROR "Message received had no Opcode" + opCode  ENDSWITCH  IF msg IS NOT NULL THEN  IF server IS NOT NULL THEN  CALL msg.ReceivedOnServer WITH cnn  ELSE  CALL msg.ReceivedOnClient  ENDIF  ENDIF  ENDPROCEDURE  Declaration of Client and Server Actions for different message types  ENDCLASS |

**Network Manager:**

The network Manager is constructed to handle initialisation of server hosting and managing client connections.

The Network Manager was a made for both the client and server. On one end, the component allows server to host allowing sessions to be held, and on the other, it allows the clients to easily join the server to play a match. The decision to combine both the client’s ability to join and the server’s ability to host within a single component was made to streamline the networking experience and provide a cohesive interface for all networking actions within the game.

|  |  |
| --- | --- |
| Justification of Single Component | Description |
| Cohesion and Clarity | Placing both client and server functionality within the same NetworkManager creates a clear, central point for all network-related activities. This approach is intuitive for developers working on the game and for players navigating the game's interface. |
| Ease of Maintenance | This subroutine manages the client-side process for joining an already hosted game session. When activated, it prompts the user to input the server's IP address. The entered IP address is then utilized to establish a connection to the server, facilitating the client's participation in the networked game environment. |

Design of Network Manager:

The Network Manager is structured comprising 2 main subroutines that are directly tied to user actions in the network joining page.

|  |  |
| --- | --- |
| Actions | Description |
| OnOnlineHostButton() | This subroutine is triggered when the user chooses to host a game. It sets up a listening point on a designated port (here, the port number is 8007) to accept incoming network connections from clients. This is effectively preparing the game to accept other players who want to join the hosted session. |
| OnOnlineConnectButton() | Subroutine handles the client-side operation to connect to a hosted game. It prompts the user to input the host’s IP address, then uses this information to attempt a connection to the server. |

Pseudocode:

|  |
| --- |
| CLASS NetworkManager  SUBROUTINE OnOnlineHostButton() #Whilst the subroutine is present on the client side script but this will not be connected to an button, therefore, client will not be able to host any games  CALL Server.Init(8007)  ENDSUBROUTINE  SUBROUTINE OnOnlineConnectButton()  ipAddress ← Input()  ENDSUBROUTINE  ENDCLASS |

1. **Matchmaking Algorithm**

**The matchmaking algorithm goes as follow:**

1. Take the user’s score
2. Try to find a room that has only one player and the player’s score in the room is +/- 100 points of the user
3. IF they are able to find one THEN: Join
4. Else: Create a room

These are the main part of the algorithm is quite a simply process, however, it is important that the server can keep track of the players within each room.

**Pseudocode:**

|  |
| --- |
| SUBROUTINE AssignPlayerToRoom(connection: NetworkConnection)  playerid ← connectionid  FOR EACH room IN rooms  roomNumber ← rooms dictionary's Key  roomPlayers ← list of players within the room number  # Check if there's only one player in the room  IF LEN(roomPlayers) ← 1 THEN  ecisitingPlayerid ← roomPLayer[0]  IF existingPlayerId in UserInfo THEN  existingPlayerScore ← exisitingPlayerId score  scoreDifference ← ABS(existingPlayerScore - joiningPlayerScore)  # Check if the score difference is within the acceptable range  IF scoreDifference <= 100 THEN  # Add player to the room  ADD playerId to roomPlayers  OUTPUT "Player " + playerId + " joined room " + roomNumber  # If the room is now full, create a GameRoom instance  gameRooms[roomNumber] ← NEW GameRoom(roomPlayers)  OUTPUT "Game can start in room " + roomNumber  roomCount ← roomCount + 1  RETURN  ENDIF  ENDIF  ENDIF  ENDFOR  # If no suitable room is found, create a new room  rooms[roomCount] ← NEW LIST OF INTEGER {playerId}  OUTPUT "Player " + playerId + " created a new room " + roomCount  roomCount ← roomCount + 1  ENDSUBROUTINE |

1. **Validation win theory:**

The initial thought to validation will be checking every single column, row and layer for a connect 4.

In fact, this would have unnecessary checks. In our algorithm we will make use of a pointer system to reduce the amount of check that has to be done. This pointer system iterates through the board, through a particular direction (i.e from left to right). This means that there will be certain regions that does not need to be checked, called boundary condition, as it is impossible to create a connect 4.

Explanation:

Consider a 5x6 board and the programme:

Note: Throughout the whole explanation section “X” are used to show the orientation of the check that I am describing.

**Horizontal Checks**

In the horizontal check, the pointer starts from (0,0) and scans horizontally left to right. During each cycle, the pointer accesses the current box and the next 3 adjacent boxes on the right. If a connect 4 is detected then there is a winner, else the pointer then moves one to the left.

If this approach is followed, your initial thought potentially think that you will need to place the pointer at each coordinate. However, if you consider placing a pointer at (0,4) and scan the adjacent 3 boxes on the left. You can see that a connect 4 will never be produced as there are not enough adjacent boxes to produce a connect 4.  
  
Therefore, as a result, there can be boundary conditions to be set on the horizontal checks – only iterating through column 0 to 3.

Horizontal check (red region represent the region that does not have to be iterated through)

A grid with red and black x

Description automatically generated

**Vertical Check**

The vertical checking algorithm operates on the same principle as the horizontal checks. Starting from the topmost row (0,0), the pointer scans downwards, checking the current box and the 3 blocks directly below it. The pointer advances one position below, if connect 4 does not get detected.

Similarly to horizontal check, boundary conditions will have to put in place in order to exclude unnecessary checks. Boundaries condition is placed so that it only iterate through row 0 to 2.

Vertical Check (red region represent the region that does not have to be iterated through)

A grid with red and white lines

Description automatically generated

**Diagonal Check:**

The diagonal check is split into two distinct iterations: a right to left, and a left to right diagonal scan. This approach is critical because a single – direction check would not cover all possible winning combinations.

Left to Right:

The pointer starts from (0,0). In each cycle, it checks the current box and ascending 3 diagonal pieces down on the right. The pointer advances one position to the right, if there is no connect 4.

In terms of boundary condition, many of the rows and columns, in fact, does not have to be iterated through. Only a small area rectangular box colour white are coordinates that the start point be placed on.

A grid with red squares and black x

Description automatically generated

Right to Left:

The pointer starts from (0,6). In each cycle, it checks the current box and ascending 3 diagonal pieces down to the left. The pointer advances one position to the left, if there is no connect 4.

Boundary condition for this iteration is virtually identical with the left to right iteration, but instead the iteration area is on the top right-hand side.

A grid with numbers and x

Description automatically generated

Implementation:

After demonstrating for the pointer system can reduce the numbers of iteration, we now have to put it into the 4x4x4 board is exactly the same. The only thing is that instead of having 3 checks, the game will have 7 checks.

An overview of all checks:

2D – involving only one layer:

* x-axis check
* z-axis check
* z-x-axis check

3D – involving multiple layers:

* y-axis check
* y-z-axis check
* x-y-axis check
* x-y-z-axis check

**2D checks:**

This is very similar to the demonstration of the 5x6. In the game, in 4x4x4 board even less iterations must be done. These checks will be done across ***each layer*** of the board. Therefore, for all the checks below, you can imagine it as if it is a 4x4 board.

x-axis check:

On the horizontal checks, it only has to iterate through column 0. As I explained in the example, since it iterates through from left to right, placing a pointer in column 1 or beyond in the board will be unnecessary, as it will be impossible to make a connect 4.

A grid with red squares and black x

Description automatically generated

z-axis check:

Similarly, on the vertical check, only the first row has to be iterated through. The idea here is the same, since the check is iterating from top to bottom. If a pointer is placed row 1 or beyond, it is impossible to make a connect 4.

A screenshot of a grid

Description automatically generated

z-x-axis check:

Pointers are only placed in (0,0) : left to right iteration and (3,0) : right to left iteration.

You might question, why I have not put pointers at (3,0) or (3,3). It is because the algorithm’s aim is only to check for a connect 4 not to find the next best move, so it does not need to identify how the connect 4 is made whether it starts. Hence, only 2 pointers in the opposite side are sufficient enough.

A grid with black x and red squares

Description automatically generated

A screenshot of a game

Description automatically generated

**3D Checks:**

After showing you how much processing time is reduced with a validation algorithm. Now, I will show you how this was implemented into the game. Unlike the example above, the game has one extra dimension (layer aspect).

Throughout the explanation I will be using the y x z direction of unity as the standard

A screenshot of a computer

Description automatically generated

If a reference of a position is every made in the form of [i, j, k], think in a form of [layer, row, column]

In the below demonstration, I will be using a prototype that I have built of the game board in unity to demonstrate as it makes it easier to visualise. I will provide the bird eye view of the board to explain the checks.

y-axis check:

In the y-axis check, there will be no boundary conditions as there will not be one row that does not have to be checked. As a result, each column in each layer must be checked vertically up. The picture below highlighted in green must be checked through.

A white square with green squares

Description automatically generated

z-y-axis check:

If you consider slices the bits horizontally – strips across the page. These strips are checked in zy-plane at each position along the x-axis. The function iterates through each column along the z-axis, increasing the y value by 1 each time within the 3D gird space and performs the following check:

1. **From left to right:** For each z-indexed column, the function examines a diagonal starting from the left most position to the right most position (considering the bottom layer as y = 3 and column z = 0, given zero-based indexing) and extends towards the right position of that column. This check is performed by comparing the values at grid positions [3, j, 0], [2, j, 1], [1, j, 2] and [0, j, 3] where j represents the x-indexed row. If all four positions hold the same non-zero value, a winning condition is detected for that diagonal.
2. **From right to left:** Similarly, the function also checks the other way around within the same z-indexed column, starting from the right-most to left-most position (y = 3 and z = 3) and proceeding towards the left-most position. The values at grid positions [3, j, 3], [2, j, 2], [1, j, 1] and [0, j, 0] are compared, and if they are identical and non-zero, it signifies a winning condition for that diagonal.

In the picture below the red highlighted bits are the boundary condition. Those are the ones that you do not need to check for. As it is impossible from left to right or right to left to create a connect 4 horizontally.

More importantly, with the boxes highlighted in green. They only must do one of the following checks. With the left most column doing from left to right, and the right most column doing from right to left.

Lastly, the zy-plane check only has to be done from the bottom most layer as it is impossible to make a connect 4 upwards from the any other layers.

A white square with red and green squares

Description automatically generated

x-y-axis check:

This check is very much like the z-y-axis check. However, this time imagine slicing the board vertically - strips down the page. These strips are checked in xy-plane at each position along the z axis. The function iterates through each row and increase the y value by one each time along the 3D plane and performs the following check:

1. **From front-most to bottom-most of each stirp:** For each x-indexed column, the function examines a vertically starting from the front-most position to the bottom-most position (considering the bottom layer as y = 3 and column x = 0, given zero-based indexing) and extends towards the right position of that column. This check is performed by comparing the values at grid positions [3, 0, k], [2, 1, k], [1, 2, k] and [0, 3, k] where k represents the z-indexed row. If all four positions hold the same non-zero value, a winning condition is detected for that diagonal.
2. **From bottom-most to front-most of each strip:** Similarly, the function also checks the other way around within the same x-indexed row, starting from the bottom-most to top-most position (y = 3 and x = 3) and proceeding towards the front-most position. The values at grid positions [3, 3, k], [2, 2, k], [1, 1, k] and [0, 0, k] are compared, and if they are identical and non-zero, it signifies a winning condition for that diagonal.

Notice in the picture before this check virtually identical to the z-y-axis check, the logic is rotated 90 degrees. The boundary are the like the z-y-axis check being the centre ones not being check in each strips.   
  
Furthermore, it is the same as z-y-axis check as each box only have to complete one of the checks given above. The front-most row checking from front-most to bottom-most; the bottom-most row checking from bottom-most to front-most row.

Lastly, x-y-axis check also only needed to be done on the bottom-most layer with the same reason of why z-y-axis is only done only on the bottom-most layer.

A white square with red and green squares

Description automatically generated

x-y-z-axis:

The last check that I must introduce to you is the x-y-z-axis check. Where there will not be any slices as it is checking diagonally across the 3 dimensions. In hindsight, it may seem that there is a lot of boxes to check. However, in fact for this check there is only 4 total checks that has to be done.   
  
Firstly, consider the y-axis, like all the other checks that are 3 dimensions, only the bottom most layer has to be checked due to the nature of a 4x4x4 - it is impossible to make a connect 4 from any other layer. As a result, I can eliminate the top 3 layers.

Secondly, consider the x-axis, in the x-axis in the middle two rows cannot make any diagonals with the reason that it is impossible to place 4 blocks diagonally from each of the boxes in the middle row without going out of bound, thus I can eliminate the middle two rows.

Lastly, consider the z-axis, in the z-axis the middle two columns cannot also make diagonals from those positions with the same reason as x-axis. With that I again can eliminate the middle two column.

With that said, at the end, this created a boundary condition for me – leaving me with only having needed to check the 4 corners of the board (highlighted in green) and the rest not need to be checked (highlighted in red).

A white square with red and green squares

Description automatically generated

## UI design

### Main Menu Interface

#### Before Login

A close-up of a login screen

Description automatically generated

Options on the screen:

* Register : Active
* Login : Active
* Local Game : Active
* AI Mode : Active
* Online Game : Disable
* Replay : Disable

In the proposed interface, the user has yet to log in and, therefore, no user-specific information can be displayed. Given that users typically read from top to bottom, it's crucial to maintain a logical sequence in the arrangement of options. The interface is designed to guide the user naturally from "Register" to "Login," which are the initial steps a new user would take.

The layout intentionally varies the spacing between options to create distinct categories. For instance, "Local Game" and "AI Mode" are placed closely together to signal to the user that these are single-player modes. This thoughtful categorization aids in distinguishing between offline and online gameplay options, making the interface intuitive and user-friendly.

Lastly, the decision to display but disable the "Online Game" and "Replay" options prior to login is deliberate. It serves to preview the full range of available features, enticing users with the possibilities that await post-login. This approach aims to retain user interest by showcasing the game's offerings, rather than limiting perceived options and potentially diminishing the game's appeal.

#### After Login

A screenshot of a login form

Description automatically generated

Display/ Option:

* Username is displayed
* Score is displayed
* Local Game : Active
* AI Mode : Active
* Online Game : Active
* Replay : Active

After the user has logged in, “Online Game” and “Replay” is now active for the user to use alongside with the already available “Local Game” and “Replay”. Furthermore, the names and the score of the username is displayed. In many games is very common for the username to be displayed on the top of all the options, thus for familiarity, I have designed so that the user information is placed on the top. Other than the name and the score, I tried to maintain a similar format as before the user login, so that there is not too much of a shock to the new user who have been recently register who might have just recently registered.

### Login Interface

A screen shot of a login form

Description automatically generated

The login page adheres to the widely recognized convention of account access, a design that users will find immediately familiar. This familiarity is key to ensuring a frictionless user experience, allowing users to log in smoothly and efficiently.

The page is straightforward, featuring a clear, uncluttered layout with labelled fields for "Name" and "Password," followed by a prominent "Submit" button. This simplicity minimizes cognitive load, allowing users to quickly understand and navigate the login process without confusion.

An additional thoughtful touch is the inclusion of an "Incorrect Password or Username" message. This immediate feedback is crucial for troubleshooting, guiding users to rectify their login details without delay. By aligning with established login procedures found in other games and applications, the design of this page provides a seamless transition for users into the gaming environment.

Top of Form

Bottom of Form

### Register Interface

A screen shot of a login form

Description automatically generated

The registration page is designed to be intuitive and user-friendly, mirroring the straightforward approach of many standard sign-up interfaces.

The form contains fields for "Name" and "Password," each accompanied by a text box, a common layout that users are accustomed to. This familiar format ensures that new users can navigate the registration process effortlessly.

The "Submit" button is prominently displayed, indicating the next step in a clear and concise manner. The registration page also provides immediate validation feedback with messages like "Username already exists" or "Password is too short." Such messages are essential as they inform the user of issues in real-time, enabling them to correct their entries promptly.

### Game Interface

The Game UI is essential in the project as it is where the user interact the most. It is important that the Game UI is effectively communicating the current state of the game and players turns – so that the users know who they are playing against and that who turn it is.

There is slight difference between the Local Option (AI Mode and Local Game) and Online Option. Therefore, I will be describing them in two parts.

#### Online:

A diagram of a computer program

Description automatically generated

In the interface, I try to emulate the minimalistic design of Lichees. It features a clear distinction of the White Player and Black Player, displaying which role they have and their username and score. This aligns with the classic setup of two-player games and allows for immediate recognition of each player’s status.

Another notable aspect of the interface is the turn indicator, displaying either “Your Turn” or “Opponent Turn”. This is crucial as it keeps the players informed about the game’s progression.

User’s engagement with the game is crucial, therefore, I introduce an interactive element that allow players to rotate the camera and view the board from different perspectives. This feature allows users to look at the board in different perspective.

#### Local Game/ AI Mode

A drawing of a cube with text and blue writing

Description automatically generated with medium confidence

In this mode, unlike the online interface, I removed the display on the top of the page of the names as there is no names or score to be displayed. This aligns with the nature of local or AI games where the focus is on game itself rather than the individual identities of the players. As a result, the turn indicator has also been adapted to fit the local gameplay context. It alternates between displaying “Black’s Turn” and “White’s Turn”, providing a clear and straightforward indication of whose move it is. This simplicity is key in a single device gameplay setting, where player is physically present, and the game does not need to track personal scores or identities across a network. Other than these differences, all the other display will be the same as the Online Interface.

### Endgame Interface

A diagram of a diagram of a game

Description automatically generated with medium confidence

Options:

* Rematch
* New Game
* Exit

This is the Endgame interface provides the users clear options following the end of the game. The Interface also display the winner of the game displayed message of “Black Wins” or “White Wins” – allowing players to easily see the outcome. This straightforward indicator is very standard in any turn base game making the interface familiar to the user.

Below the outcome, the players are presented with three distinct choices to proceed: they can select “Rematch” to engage in another round with the same opponent, when the user click the button, it will indicate to the other user with the message “Opponent want rematch” to indicate that the user is asking for a rematch. Then there is “New Game” which allows the user to join a new game. Lastly, the “Exit” option is provided if the suer want to leave the online game environment, this returns them back to the main menu. With “New Game” and “Exit” once pressed, the opponent will be indicated with the message “Opponent has left” indicating that they do not want rematch.

### Replay Interface

#### Menu

A screenshot of a computer screen

Description automatically generated

Upon selecting “Replay” from the Main Menu, users are presented with a list detailing their 10 most recent games. For each gale listed, the interface clearly displays the opponent’s name, the game’s result, the total number of moves played, and a button called “Go To Game”. The button stands out as a direct call-to-action, eliminating any guesswork for the user on how to access the game replay, thus enhancing the user experience through clear navigation.

On the right-hand side of the interface, the presence of the scrollbar is a subtle design choice. It indicates to the user that the list is scrollable, extending beyond what they can see. Furthermore, it prevents the user to be overwhelmed with the amount of information in front of them, as this allow the information to be more spaced out and not overload the screen.

#### Game Board

A drawing of a computer program

Description automatically generated with medium confidence

After clicking on the “Go To Game”, the user is then transferred to this page where the user will be able to view the game that they have picked. The focus of the replay mode is the game. There ensuring that the design is simple and that prioritising clear visibility of the game is key.

The associate Black and White Player’s username and score is displayed above the board. T Winner name is also distinctive displayed below the board to reduce the chance of confusion in terms who have won the game. The board continued to be able to be rotated around allow the user view different perspective by using the mouse. The use of mouse is a convention use by game developers to rotate around an object, this familiarity will reduce any confusion. Furthermore, you notice that there are also forward and backward buttons that can be clicked, these are good as they reduce any ambiguity with users on where to click to progress in state. However, with that being said, the game also allow user to use keyboard to control the state. Having two options allow users to choose giving them a greater freedom of choice.

### Online Interface

#### Joining Menu

A whiteboard with a login screen

Description automatically generated with medium confidence

Upon clicking on Online in the Main Menu, the user will be brought to this screen. The design of the online menu is purposefully minimalist and user – centric, adhering to established gaming convention to ensure familiarity and ease of user.

It has been structured to facilitate a seamless and intuitive connection process for the user. As the user navigate to this menu, they find an interface where they can easily input or confirm the IP address they wish to join – the default being set to 127.0.0.1, which serves as a starting point for users looking to connect to a game server. Once the users decided which server to join, they click onto the Connect button to send a request to join a match. The button is positioned centrally, making it clear for users wishing to initiate a connection. In addition, a Back Button is present on the screen, this button is positioned to suggest a secondary action, consistent with user interface design. This button allows users to return to the main menu. This is placed to ensure that users who wish to return are given the option to do so.

#### Waiting Page

A whiteboard with writing on it

Description automatically generated

The waiting page for the online menu is crafted to keep users informed while they anticipate the establishment of a connection for an online match. It’s simple yet essential part of the user experience.

Upon this page, users are greeted with a clear message indicating that the system is in the processed and they need to wait for just a little longer. If there is a change of mind or a decision not to continue waiting, the Exit button is prominently displayed, offering users an easy way to withdraw from the waiting screen and return to main menu. This design considers the user’s autonomy and necessitating additional actions or causing confusion.

### AI Interface

#### Difficulty Menu

A whiteboard with a box with text and arrows

Description automatically generated with medium confidence

After clicking on AI Mode in the main menu, user will be brought to this page to select the level of AI they wish to play. This menu is designed to be straightforward and focused, minimising distractions and ensuring that players can quickly and easily set the game to a level that matches their skill and desired challenge.

There are 3 options – Easy, Medium and Hard. These universally understood terms provide immediate insight into the nature of the challenge players can expect from the AI. Such clarity is essential in allowing players to make an informed choice that aligns with their gameplay preferences or the level of difficulty they feel prepared to engage with.

The Exit button routes the user back to the Main Menu, presenting an option for users who wish to return to the main. The button is placed away from the different options of difficulty to show the distinct two options the user have – to play a match or to leave.

## API

There are 3 situations in this game where the user need:

1. Login
2. Registration
3. Replay Information Retrieval
   1. 10 Most recent games
   2. Moves of the selected game

Login:

This API handles user authentication. Upon receiving the username and password from the client, the server hashes and salts the password, then compares it with the stored hash in the database for validation.

Registration:

This API facilitates the creation of new user accounts. It generates a unique identifier for the user, hashes and salt the password, and stores user information in the database.

Replay Information Retrieval:

This process is divided into two APIs. Firstly, the client requests the ten most recent games. Then, based on the selection made by the client from the list, another request is sent to retrieve the moves associated with the selected game. This separation optimises data retrieval, ensuring that moves are fetched only if the user intends to replay a specific game.

In our game all our API requests are done via the use of PHP and involves a POST request. The server executes SQL queries based on the request to interact with the database.

Table of the different API

|  |  |
| --- | --- |
| Type of API | Description |
| Login | Handles user authentication and return user score and ID. |
| Register | Facilitates user registration and stores user information |
| Get Moves | Retrieves move associated with a specific game session |
| Retrieve Recent Game | Retrieves information about the ten most recent games played by a user. |

Procedure of each API call:

Login API:

1. Retrieve username and password from POST request.

2. Connect to the database.

3. Hash and salt the received password.

4. Execute SQL query to fetch stored hash for the provided username.

5. Compare the hashed password from step 3 with the stored hash.

6. If passwords match, return success message; otherwise, return an error message.

7. Close the database connection.

Registration API:

1. Retrieve username and password from POST request.

2. Generate a unique identifier for the new user.

3. Hash and salt the received password.

4. Connect to the database.

5. Execute SQL query to check for duplicate usernames.

6. If no duplicate found, insert the new user information into the database.

7. Return success message upon successful registration.

8. Close the database connection.

Replay Information Retrieval API (Part 1 - 10 Most Recent Games):

1. Connect to the database.

2. Execute SQL query to retrieve the ten most recent games.

3. Return the game information in JSON format.

4. Close the database connection.

Replay Information Retrieval API (Part 2 - Moves of the Selected Game):

1. Retrieve the selected game ID from the POST request.

2. Connect to the database.

3. Execute SQL query to retrieve the moves associated with the selected game.

4. Return the moves information in JSON format.

5. Close the database connection.

## Database

After discussing about the different API calls the client can make. I have to now discuss the design of the database.

### Initial design

**Entity Relationship Diagram**

**A diagram of a function

Description automatically generated**

### Entity Attribute Models

**Normalisation**

Database Design:

Below, I ran through the steps that I took to Normalised my database.

Unnormalised Data



**Stages:**

First Normal form (1NF):

Objective: Ensure that a table does not contain repeating attributes or repeating groups and that all the data in the table is atomic



The initial dataset contained multiple values in single fields and groups of data that repeated across different rows, making it difficult to update or delete records without potential data loss. Also, if a user plays more than one game then there will be multiple records for the same user.

To address this, I have broken down the fields to ensure that each one contains the smallest possible unit of information, and I have separated the repeating groups into new tables of User Table, Session Table and Transaction Table.

Second Normal Form (2NF):

Objective: Ensure the database is in 1NF and then removing attributes that depend upon parts but not all the primary key by creating additional table





Upon reaching the 1NF, I noticed that attributes of Black Player ID (B\_Player) and White Player ID (W\_Player) were identified as being partially dependent on the primary key of UserID, which was not desirable as it could lead to redundancy and maintenance issues. To resolve this, a junction table was introduced, which is specifically designed to handle many – to – many relationships. This new table, called “Game\_UserID”, includes Session ID, UserID, Colour, creating a composite primary key that encapsulate the many – to many relationships between the User Table and Session Table by having two One – to – many relationship [Reference to the Entity Relation Table below]

Moreover, Score from User Table stood out to me. It was clearly storing scores directly within the User Table would lead to unnecessary duplication since scores could be calculated from the Transaction Table. This duplication could result in inconsistencies, especially if the score calculation logic ever changed.

3NF:

Objective: Ensure the database is in 2NF and then removing non – key attributes that depend upon other non-key attributes by creating additional table or assigning foreign key.



I ensured that all tables did not have any transitive dependences. I wanted to guarantee that all non – attributes were only dependent on the primary key. For Game\_UserID Table, I realise it was necessary to assign foreign keys for SessionID and UserID, which makes up the composite Primary Key. This ensure that there was no indirect relationship or derived data within the Game\_UserID.

I also assigned WinnerID (in Session Table) and (SessionID and UserID) as foreign keys to also ensure that there was no derived user data within the constituent tables.

### Final Database

A screenshot of a computer

Description automatically generated

## SQL Statement

With the API calls, these are the following SQL statement that I have designed for the server return the data needed by the client.

### Save Game

Firstly, I will show you the queries done by the server involving with the save of the game state after the result have been determined.

|  |  |  |
| --- | --- | --- |
| No | Description | Purpose |
| 1 | Insert Moves and Winner to Session Table | This query inserts the winner’s ID and the moves made during the game into the Session table |
| 2 | Insert UserID and SessionID in Game\_UserID Table | This query associates a user with a game session by inserting a user ID and session ID into the Game\_UserID junction table. |
| 3 | Insert into Transaction Table | This query logs a transaction, where when a game finishes, it is executed. Depending on the result of the game, the values used will be different – winner : + 10, loser : -10 , draw : 0. |

1.Insert Moves and Winner to Session Table

|  |
| --- |
| INSERT INTO Session (WinnerID, Moves) VALUES (?, ?); |

2.Insert UserID and SessionID in Game\_UserID:

|  |
| --- |
| INSERT INTO Game\_UserID (UserID, SessionID) VALUES (?, ?); |

3.Insert into Transaction Table:

|  |
| --- |
| INSERT INTO Transaction (SessionID, UserID, PointChange, Reason) VALUES (?, ?, ?, ?); |

### 

### Register:

This process is split into 2 queries:

|  |  |  |
| --- | --- | --- |
| No | Description | Purpose |
| 1 | Selects the username from UserID table if the username inputted is equal to any username in the table | Check for Username Availability, preventing any duplication |
| 2 | It essentially adds the user information into the database UserID table – adding their username, hash, salt. | Create User Account.  The reason why the userId itself does not need to be generated is because the userId element is set as incremental. This also solve the issue of userId uniqueness as the userId will not always be unique |

1.Check for Username availability

|  |
| --- |
| SELECT username FROM UserID WHERE username = name; |

2.Create User Account

|  |
| --- |
| INSERT INTO UserID (username, hash ,salt) VALUES(?,?,?); |

### Login

This process online involves one query to authenticate a user attempting to log in. This query is used to retrieve the userID, salt and hash or a username during the login process. This information is used to verify the user’s password. While this validation logic is not done here, the information is used for the API request.

|  |
| --- |
| SELECT id, salt, hash FROM UserID WHERE username = '$username'; |

### Retrieve Data for Replay

This query is executed when the user clicks on “Replay” on the main menu. The user passes through their userId into the query, it then retrieve information about the 10 most recent game sessions for the specific user - returning the opponent score, the winner’s username, the role of the user, and the total number of moves in the game.

In order to retrieve these data, the following steps are taken:

1. **Identify User's Recent Games**:
   * I need to access the game sessions associated with a specific user. This means I will need the user's ID to filter the game sessions. I started by looking into the Game\_UserID table, where game sessions are linked to user IDs.
2. **Get the Most Recent Sessions**:
   * I want to order the sessions by their session ID in descending order to get the most recent first. This can be done by using ORDER BY clause on SessionID in a descending (DESC) fashion.
   * Since I am interested in only the 10 most recent sessions, I added a LIMIT 10 clause at the end of our query.
3. **Join with Game Information**:
   * I need additional details from the GameInfo table, like the ID of the game's winner and the moves made during the game. To get this, I performed an INNER JOIN on the GameInfo table using SessionID as the common column to link the data.
4. **Retrieve Opponent's Information**:
   * To get the opponent's username and calculate their score, I needed to join with the UserID table, which contains the usernames. Since each game involves two users (the current user and the opponent), I ensured that we don't retrieve the current user's username as the opponent's. This involved a condition to exclude the current user's ID.
5. **Calculate Opponent's Score**:
   * The score is calculated by starting with a base score of 1000 and adding the sum of points changes from the Transaction table. This requires a subquery that selects the sum of PointsChange from Transaction where the UserID matches the opponent's ID.
6. **Determine the Winner**:
   * To know who won each game. This involves another subquery that retrieves the winner's username from the UserID table using the WinnerID from the GameInfo table.
7. **Identify the User's Role (Colour)**:
   * The game's outcome may depend on whether the user played as Black or White. We need to use a CASE statement to translate the Colour column into a user-friendly role name.
8. **Count the Total Moves**:
   * To know how many moves were made in each game. Assuming the moves are stored in a JSON array in the Moves column of the GameInfo table, we can use a function like JSON\_LENGTH to count them.

Final Result:

|  |
| --- |
| SELECT  gi.SessionID,  opp.username AS OpponentUsername,  (1000 + IFNULL((SELECT SUM(t.PointsChange) FROM Transaction t WHERE t.UserID = opp.id), 0)) AS OpponentScore,  (SELECT username FROM UserID WHERE id = gi.WinnerID) AS WinnerUsername,  CASE  WHEN gu.Colour = 1 THEN 'Black'  WHEN gu.Colour = 2 THEN 'White'  END AS UserRole,  JSON\_LENGTH(gi.Moves) AS TotalMoves  FROM  Game\_UserID gu  INNER JOIN GameInfo gi ON gu.SessionID = gi.SessionID  INNER JOIN UserID usr ON gu.UserID = usr.id  INNER JOIN UserID opp ON opp.id <> usr.id AND opp.id IN  (SELECT UserID FROM Game\_UserID WHERE SessionID = gi.SessionID AND UserID <> gu.UserID)  WHERE  gu.UserID = ?  ORDER BY  gi.SessionID DESC  LIMIT 10; |

### Retrieving Moves

Once the player has clicked on “Go To Game” this query is executed by in order to retrieve the moves of the game using the sessionID. The process of the query is very straightforward, it goes into the Session table and go through the column of SessionID until the sessionID found is equal to the session ID inputted in “?”.

|  |
| --- |
| SELECT Moves FROM Session WHERE SessionID = ? |

Data Serialisation and Deserialisation:  
In the server, the moves were stored in a list. However, since list is not a type that can be stored in a database, as a result, in order to solve this problem the list is converted in the json string via the process or serialisation. When the moves have to be retrieved back for replay mode, the string is then deserialised which convert to a list that can be iterate through by a pointer in replay mode.

Serialisation:

Serialisation is the process of converting the state of objects within my game into a format that can be easily stored or transmitted. This significantly comes into play in the context of my Replay class, when a game concludes, the sequence of moves made by the players – represented in your program as a list of Vector3Int coordinates – needs to be serialized into a JSON string. This allows the move data to be stored in a standardized and efficient format, whether it's in a local file, a database, or sent over the network to a server.

Benefits of Serialization:

* Persistence: Serialization allows game state to be preserved between sessions.
* Interoperability: JSON, as a text-based format, can be read and processed by various systems and programming languages.
* Data Integrity: Encapsulating the game state in a structured format like JSON helps ensure that all necessary data is included and correctly formatted.

Deserialisation:

Deserialisation is the reverse process, where you tale structured data and reconstruct it into objects or data structure that my game can utilise. When initiating a replay, the JSON string containing the move data is deserialised back into a list of Vector3Int coordinates. As a result, my Replay class can then user this deserialised data to reenact the moves on the game board.

Benefit of Deserialisation:

* Game Replay: Allows the recreation of pass games for player review.
* Data Recovery: Converts the stored data back into a useable in – game format.
* Flexibility: Supports modifications in how moves are processed and replayed without changing the stored data format.

Pseudocode of Serialisation

|  |
| --- |
| SUBROUTINE SerializeToJson(vectorList)  listOfLists ← NEW List of List of integers  FOR EACH vector IN vectorList  tempList ← NEW List of integers  ADD vector.x TO tempList  ADD vector.y TO tempList  ADD vector.z TO tempList  ADD tempList TO listOfLists  END FOR  json ← CONVERT listOfLists TO JSON STRING  RETURN json  END SUBROUTINE |

Pseudocode of Deserialisation

|  |
| --- |
| SUBROUTINE DeserializeFromJson(json)  listOfLists ← CONVERT json STRING TO List of List of integers  IF listOfLists IS NOT NULL THEN  FOR EACH coord IN listOfLists  IF SIZE OF coord EQUALS 3 THEN  vector ← NEW Vector3Int WITH coord[0], coord[1], coord[2]  ADD vector TO moves  END IF  END FOR  END IF  END SUBROUTINE |

# UML

This is printed onto a A3 Sheet of paper as it was too big to fit onto the page.

# Technical Solution

Technical Solution is within the Appendix

|  |  |
| --- | --- |
| Aspects | Page Number |
| Code | 137 – 222 |
| Function Table | 223 - 229 |
| Data Table | 230 - 233 |

# Testing

All Testing that were done are recorded into a YouTube video.

Please follow this link to see: <https://youtu.be/7GjDPE9J_ao>

Furthermore, testing plan is printed on an A3 Sheet of paper as it was too large to fit onto the page.

# Evaluation

Throughout the project, my aim was to create an innovative take on the classic game of connect 4, by introducing a 3D gameplay environment that enhances the traditional experience. This journey taught me many new things, including learning a whole new type of programming language, database design, the foundation of AI algorithm and networking for multiplayer functionality. However, the most notable thing that the NEA project has taught me was putting user’s experience and useability as priority.

My objectives were to make use of Unity software to build a game that is both engaging and challenging, while also incorporating a robust multiplayer feature through a client-server model. I sought to not only fulfil the technical requirement, but also learn something from the project that I chose.

### Comparing performance against the objectives

|  |  |  |  |
| --- | --- | --- | --- |
| No | Objective | Performance Criteria | Evaluation |
| 1 | Design the 3D Connect 4 game board logic for a 4x4x4 grid. | The logic must accurately track each move the board state, validate the legality of moves, and enforce game rules. | Completely achieved:  The logic of the 3D Connect 4 game works completely; the system can check for every possible connect 4 that can be met. To track the board state, I implemented a pre-initialized 3D array to represent the board. This is then manipulated through the game to do various checks. To validate the legality of moves, I used the 3D array and checked whether that specific x-z coordinate still have a free layer. If the x-z coordinate has reached the max and has already been filled up, then it is full; else I just return the y value of the next free space up the board. Furthermore, to ensure that the game rules are enforced throughout testing, I implemented an additional debug board function which outputs a string version of the 3D array (which represents the board), allowing me to very easily spot if functionality relating to the gameplay is not working.  Furthermore, to ensure that the game rules are enforced throughout testing, I implemented an addition debug board function which output a string version of the 3D array (which represents the board), allow me to very easily spot if a functionality relating to the gameplay is not working. |
| 2 | Develop the graphical user interface for the 4x4x4 3D Board. | The GUI should clearly display the grid and differentiate between empty and filled spaces with high contrasting visual appeal. | The UI design, subjective by nature, was successfully executed. A 3D Board was created in Blender, offering clear distinction for block placement possibilities. This being the first experience with Blender, the result was satisfactory and met the project's visual requirements.  As user experience is my main priority, as a result, I will be conducting an interview with the end user to see their option on this matter. |
| 3 | Implement block design with distinct contrasting colours. | Blocks must have visually distinct colours or patterns that are consistent throughout the game to avoid confusion. | Completely achieved:  Blocks designed in Blender have been implemented with stark contrasting colours—black and white—to ensure clear distinction between player moves. This contrast was maintained consistently throughout gameplay and was verified throughout the testing phase. |
| 4 | Develop user-controller board rotation functionality. | The board should rotate smoothly with intuitive controls, maintaining the player’s orientation and game focus. | Completely achieved:  A system was developed allowing users to smoothly control the camera around the board without disrupting gameplay. Limitations were set to prevent the camera from moving into disorienting angles, thus enhancing the board rotation functionality. |
| 5 | Create interactive real-time block highlight system | As the user navigates, the system should highlight the column clearly without obstructing the gameplay. | Completely achieved:  The system I built enables users to highlight the different potential position the want to place the block without distributing the opponent gameplay during the game.  I used a very distinctive red colour to highlight the box so that it is very clear on the white coloured board. |
| 6 | Design a win detection algorithm for connect 4 in 3D space. | The algorithm must quickly and accurately identify winning conditions across all planes and axes and declare the winner without errors. | Completely achieved:  Enforcing game validation rule and validation win was a key in this project. It was the main part of the performance criteria, and I discussed a significant amount of this in the design section. The validation is shown to work in the testing section [No. 4]  At the end, I was able to complete the object. In addition, I found a more effective way in doing the checks rather than iterating through every single possible square, reducing any redundant checking and hugely reduced the processing time of the validation checking algorithm. |
| 7 | Establish a secure and efficient thin client server multiplayer network | The network must handle multiple games with minimal latency, ensuring game state synchronisation and user connectivity. | Completely achieved:  I built a thin client server network that enabled the client and server to communicate through their process of playing online.  To do that, I made use of Net Messages framework and Network Message Handler script to handle the incoming messages. To implement the NetMessage framework, I made a NetUtility static script which acts like a library, which enables me to predefine the different messages that are able to be sent out, and the type of data that are expected to be received. As a result, then I would be able to register different Network Handlers to the different NetMessages, so that if one of the NetMessages were to be received by the client or the server then they would know how to deal with it. With that said, the client and server have their own network handler.  As I have said, this was a thin client-server network. Therefore, most of the system logic was done within the server-side scripting (i.e. tracking network activity, assigning clients to rooms and saving data). The only thing the was on the side of the client was to drop the block and to send different messages for to the server to be processed. |
| 8 | Develop a user authentication system. | The system must use hash and salt for password to ensure user account is protected. | Completely achieve:  The system I built is password protected so that only the client would know the password to access their information within the system. Moreover, the hashing and salting also prevent any memory leak from the database to gain access to users’ account.  Most of this process is done on the server side via an API request, and this is where the hashing and salting is done.  In terms of system the user is interacting with, in the registration process, the system will insist that the password must be more than 11 characters long. This was shown in testing in No 10, where I registered and showed you that the password stored as hashed and salted in the database |
| 9 | Design a user-friendly UI for game navigation and in-game action. | The UI should allow for easy access to all features, with clear visual cues and minimal response time for actions. | This is very similar to objective 2 in the sense that is a very subjective thing. As a result, in order to see the thoughts of the end user.  For now, I have tried to create a UI that allow for easy access to all features with clear visual cues and minimal response time for actions.  I created in total 10 scenes, in which 6 of the pages are menus in order to break down the process – so that the user would not be overwhelmed with information.   Most of the menu are just a sub menu of the options that are on the main menu. In the project, the main menu is very simple and straight forward, listing out all the different options that can be chosen. This acts as the central menu option where the client get to choice different functionality and modes. |
| 10 | Construct a normalised database for storing game and user data | The database should enable efficient data retrieval and update and support complex queries for gameplay analysis. | Completely achieved:  In the design section [reference], I discussed in depth in the normalisation of the database. Going through the steps on the rationale behind my decision.  At the end, I implemented 4 table to fully normalise the database. Adding the junction table “Game\_UserID” was a very huge part in normalising the database – splitting the many to many relationships to 2 one to one relationship and creating a composite primary key with userID and sessionID. This enabled me to maximise the use of inner joins to query for certain data. This was especially shown in the query that required me to retrieve the data10 most games for the user.  Furthermore, another key part that was discovered during the process of designing the database was that I realise score, which I thought would definitely be need, was in fact not needed due to the small amount of time the server retrieves that piece of data. Instead of that, I derived the score of the user by using the transaction table – by summing up all point changes and adding that to the base score of 1000.  In addition to the junction table, I also implemented a Transaction table to keep track of the transaction history, a user table which tracks the registered user account and session table which stores the winner’s id of each session and the moves. |
| 11 | Implement SQL operations for game data retrieval and user registration. | Users should be able to register and retrieve their game data efficiently using simple SQL queries, ensuring data consistency. | Completely achieved:  The client was able to send API request to the server to retrieve their gam data efficiently using simple SQL queries. This was shown in testing process.These all showed that the server is able to successfully query and return back the relevant data to the user. |
| 12 | Enable username uniqueness check during registration. | The system must validate new usernames against the database to ensure they are already in use. A clear feedback message should be outputted to the client if the username is already used after checking. | Completely achieved:  The system I built was able to compare the new username the client wants to use against the data. This is done by client sending an API request to the server, and the server trying to find if it is on the database. Testing process [No.] shows that this was accomplished, as it tried using different types of data and it was able to create account when the username is not yet registered.  Furthermore, when the client tries to use an existing name to register, the system output a message “Username already exist” to indicate to the user. |
| 13 | Design login verification to authenticate users. | User credentials must be verified against stored encrypted data, with clear feedback provided for authentication success or failure. | Completely achieve:  The system uses the hash and the salt to verify the user inputted password.  To do so, the system first looks up the user’s record in the database and retrieves the salt that was stored during the account creation. Then, the system concatenates the user-provided password with the retrieved slat and applies the same hash function used during registration. The system compares the hash generated from the user provided password with the stored hash. If the two hashes match, the password is considered correct, and the suer is authenticated. If they do not match, then the login attempt is denied, and a message is outputted on the screen “Incorrect username or password”.  In testing, this was shown to work, the system was able to verify the user without fail. Therefore, I believe that I have achieved the performance criteria. |
| 14 | Integrate an AI opponent with varying difficulty | The AI should provide a challenge appropriate to the user’s skill level, using the Mini-Max algorithm to make strategic moves. | What define a good AI is a very much subjective thing.  If I go by the books, in terms of the algorithmic side of the AI, I will argue that it is very good.  The reason for that is because I have not only added the fundamental parts that runs the algorithm, in addition, I also added bits that makes the AI process faster. By implementing Alpha Beta pruning and Zobrist hashing, I was able to go from running the depth of 2 recursion depth maximum at the beginning without these implementations. To around 5 recursion depth.  As a result, I was able to provide a greater range of difficulty shown in the testing process. Allowing the difficulty of easy, medium and hard to be more distinctive. |
| 15 | Create a replay menu which list the 10 most recent games the player has played, including detail about each match | The replay menu should display the 10 most recent games with detailed such as opponent name, outcome and moved played. | Completely achieve:  I built a replay menu page which shows the user their ten most recent game. This is done by sending an API request to the server and executing query and returning a json string back to the user which then get deserialised. After, it is having deserialised, each game information is formatted in a way which is very readable to the user – showing them the opponent’s name, score, result and the number of moves played. In addition to that a button is created for each of the games which when clicked allow you to go and watch the replay – shown in testing process [no.].  Each game is presented in this format, creating very consistent page for the user to navigate. |
| 16 | Develop a system that allows users to replay a match from the replay menu | The user should be able to select and view a full replay of any listed match. The system should ensure the replay accurately reflects the game state transitions that occurred during the match. | Completely achieved:  the user can select from the replay menu any match they want to watch presented on the screen. |
| 17 | Implement functionality for users to control the replay of a match, allowing them to navigate moves backward and forward | Users should have controls to step through each move of the game, move by move, both backward and forward. The system should display the game state for each move accurately without any lag. | Completely achieved:  The user can navigate forward and backward without any problem shown in the testing. |

### Independent feedback

|  |  |  |  |
| --- | --- | --- | --- |
| No | Objective | Performance Criteria | Evaluation |
| 2 | Develop the graphical user interface for the 4x4x4 3D Board. | The GUI should clearly display the grid and differentiate between empty and filled spaces with high contrasting visual appeal. | “I thought the Design of the 3D board was quite nice, it was quite obvious to me at the start, straight away I knew that where I am able to place a block. Having it being white, the board pops out quite distinctively from the background dark blue, which is very good. Also, with it being right at the centre of the screen, I can tell clearly that the baord is the centre of attention. ” |
| 3 | Implement block design with distinct contrasting colours. | Blocks must have visually distinct colours or patterns that are consistent throughout the game to avoid confusion. | “The blocks colours are very distinctive, but the problem is that the board colour is clashing with the white blocks. Sometimes it can be hard to see the board and the white blocks” |
| 4 | Develop user-controller board rotation functionality. | The board should rotate smoothly with intuitive controls, maintaining the player’s orientation and game focus. | “The one thing I want to say first is that I wish there were buttons that I can use on the screen to also rotate the camera around the board. Aside from that, the rotation works quite well, I know exactly how to do use it straight away, it didn’t feel new to me, and it is good that you can see different angle of the board.” |
| 5 | Create interactive real-time block highlight system | As the user navigates, the system should highlight the column clearly without obstructing the gameplay. | “The highlight was not as smooth as I intended, I felt that it was not highlight the one I want to highlight on, maybe that is my fault. This only happens when the game lasts a bit longer and the board have more pieces, but at the start of the game it works very effectively and I like it, it pops out quite a lot showing clearly with the red”  “This was one that I thought only was not worked as well as I intended. There was a small thing about it, it is that you cannot hover over the whole column, I realise that you it only detect the column when you actually press on the box. It is not a very big thing but it would be nice if it get fixed” |
| 7 | Establish a secure and efficient thin client server multiplayer network | The network must handle multiple games with minimal latency, ensuring game state synchronisation and user connectivity. | “The network ran very smoothly during the game.” |
| 9 | Design a user-friendly UI for game navigation and in-game action. | The UI should allow for easy access to all features, with clear visual cues and minimal response time for actions. | “The in game UI is quite standard, I can tell who I am playing against and also who is white and black. Also I liked the fact that at the bottom you were told whose go it is, because there was a few times that I had to look at it to know it was my go” |
| 14 | Integrate an AI opponent with varying difficulty | The AI should provide a challenge appropriate to the user’s skill level, using the Mini-Max algorithm to make strategic moves. | “The AI is great, I felt like there was distinct difficulty changed between the three of the option, I was able to beat the first and second one, but I have to say the Hard Difficult is very much challenging and I thought that it was good. The problem I encounter though is when I play against the bot they play a move very quickly and sometimes I miss the move, so I think adding something to highlight the last move made will be a good idea” |
| 16 | Create a replay menu which list the 10 most recent games the player has played, including detail about each match | The replay menu should display the 10 most recent games with detailed such as opponent name, outcome and moved played. | “The replay menu was what I was expecting, it felt like the one that I have seen on chess.com and I personally like the design of it.” |
| 18 | Implement functionality for users to control the replay of a match, allowing them to navigate moves backward and forward | Users should have controls to step through each move of the game, move by move, both backward and forward. The system should display the game state for each move accurately without any lag. | “I liked that you gave two options to me (keyboard and button), it is really easy to use it. The UI is also quite good, you are told who you played against in that game and the result of it. One thing though, It will be good if you add something in chess.com where you can make your own move on the board and play some scenarios” |

## What would I do to further improve my Project

I believe that there are a few parts of the game that could be improved.

|  |  |  |
| --- | --- | --- |
| No | Improvement | Description |
| 1 | highlight block system | When I gave the game to different people to play the game, frequent feedback that gave me was to improve the highlighting system. They wanted it so that if you hover over the column it will highlight the whole column. This can be done by varying the box collider so that it covers the whole column. This so makes it when you hover the column it will highlight the column |
| 2 | Colour box choices | Some testers thought that the colours of the board and the block clashed slightly and would prefer another colour. To fix this, implementing a system allowing them to set their colour of their choosing. |
| 3 | Making buttons on the screen and allow users to use keyboard to rotate around the board. | To improve on my rotating camera system, I think it will be good for the user to be allowed to use the keyboard or buttons on the board to rotate around the board. This give the client more freedom on what they want to use as different users have different preferences. |
| 4 | Redesigning the UI | Redesigning the UI maybe be one of the things to do, it is a way to quickly improve the user experience by making the UI nicer. At the moment it is a bit dull and by making it slightly nicer. |
| 5 | Implementing a general and a seasonal ranking system | Ranking system was one of the implementations that was thought to be something good to have within the game shown by the survey that I conducted early in the project. When I did the project, I did not implement it initially due to the time constraints I was put on. However, it will be good to see how players are doing within the system |
| 6 | Request and add friends | Having the ability to add friends within the game will be nice and it will improve their experience. This also enable them to do friendly games instead of the random multiplayer game |
| 7 | Transferring the database into the server | I have discussed this earlier in analysis, I discussed about hosting the server on my device. However, with the development of the game, it will be better to host the server on a server making the server more stable. |
| 8 | Implementing Puzzle mode | Potentially in the game I can implement a new option which is puzzle. This is seen in many of the online boardgames like the one I analysis Lichess. I believe that this will allow the user to improve quickly. |
| 9 | Improving the scoring system | The current scoring system awards 10 points for a win and subtracts 10 points for a loss, with draws resulting in no points change. To enhance competitiveness and accurately reflect player skill, implementing the Glick rating system in the future will brings the game towards goal. The system, used by Chess.come and Lichess, incorporates factors like rating volatility and the reliability of a player’s rating, providing a more dynamic and precise measure of player’s performance. |
| 10 | Moving the server host into a dedicated server | At the moment, the server host is my computer. As a result, this makes the processing a bit slower in comparison to games that are hosted on dedicated server. I believe that for future development of the game, one key part is to find a dedicated server to host the game instead of using the desktop. |
| 11 | Adding a player manual to help player learn the game | In the future, by adding a written manual or a video explaining the various component of the game can help new player quickly immerse in the game. |

Conclusion:

My journey of creating a 3D Connect 4 game has been both challenging and enlightening, stretching my capabilities working within a time constraint and enhancing my understanding of game development. I set out to transform a classic game into a modern 3D experience.

Throughout the project, I maintained a steadfast focus on user experience and usability, anchoring every development decision to this principle. The performance of each game component against the objectives set at the beginning has been thoroughly evaluated. The results show that the core functionalities—such as the 3D game logic, user interface, AI difficulty levels, and networking capabilities—have been successfully implemented and met the criteria of providing an engaging and challenging experience for the user.

The feedback received has been invaluable, revealing insights into areas ripe for improvement. The highlighting system, colour choices, camera controls, and user interface design were all highlighted as aspects that could be refined for an even more polished experience. The potential for adding new features such as a ranking system, friend requests, and dedicated server hosting presents exciting opportunities for growth and expansion of the game's ecosystem.

## Code Appendix

### DBManager

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public static class DBManager  {  public static string username;  public static int score;  public static int userid;  public static bool online = false;  public static bool ai = false;  public static int difficulty = 0;  public static bool LoggedIn { get { return username != null; } }//if username nothing == not login  public static void LogOut() //so we can log out when we need to  {  username = null;  }  } | |

### GameUI

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | using System.Collections;  using System.Collections.Generic;  using TMPro;  using UnityEngine;  using UnityEngine.UI;  public class GameUI : MonoBehaviour  {  [SerializeField] private GameObject Auth;  [SerializeField] private GameObject LogOut;  [SerializeField] private Button registerButton;  [SerializeField] private Button loginButton;  [SerializeField] private Button OnlineButton;  [SerializeField] private Button LocalButton;  [SerializeField] private Button ReplayButton;  [SerializeField] private TMP\_Text userDisplay;  private void Start()  {  if (DBManager.LoggedIn)  {  Auth.gameObject.SetActive(false);  userDisplay.text = $”{DBManager.username} ({DBManager.score.ToString()})";  LogOut.gameObject.SetActive(true);  }  else  {  LogOut.gameObject.SetActive(false);  }    registerButton.interactable = !DBManager.LoggedIn;  loginButton.interactable = !DBManager.LoggedIn;  LocalButton.interactable = true;  OnlineButton.interactable = DBManager.LoggedIn;  ReplayButton.interactable = DBManager.LoggedIn;  }  } | |

### Login

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | using System.Collections;  using UnityEngine;  using UnityEngine.Networking;  using UnityEngine.UI;  using TMPro;  using UnityEditor.PackageManager;  public class Login : MonoBehaviour  {  public TMP\_InputField nameField;  public TMP\_InputField passwordField;  public Button submitButton;  public TMP\_Text errorMessage;  public void CallLogin()  {  StartCoroutine(LoginPlayer());  }  IEnumerator LoginPlayer()  {  WWWForm form = new WWWForm();  form.AddField("name", nameField.text);  form.AddField("password", passwordField.text);  string url = "http://localhost:8888/3D\_Connect\_4/login.php";  using (UnityWebRequest www = UnityWebRequest.Post(url, form))  {  yield return www.SendWebRequest();  string responseText = www.downloadHandler.text;  string[] responseParts = responseText.Split('\t');  if (responseParts[0] == "0")  {  DBManager.username = nameField.text;  DBManager.score = int.Parse(responseParts[1]); //force it to be a string with int.Parse  DBManager.userid = int.Parse(responseParts[2]);  UnityEngine.SceneManagement.SceneManager.LoadScene("GameUI");  }  else  {  errorMessage.text = "Incorrect Username or Password";  }      }  }  public void VerifyInputs()  {  submitButton.interactable = (nameField.text.Length >= 8 && passwordField.text.Length >= 8);  }  } | |

### Register

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Networking;  using UnityEngine.UI;  using TMPro;  public class Registration : MonoBehaviour  {  public TMP\_InputField nameField;  public TMP\_InputField passwordField;  public Button submitButton;  public TMP\_Text errorMessage;  public void CallResgister()  {  StartCoroutine(Register());  }  IEnumerator Register()  {  WWWForm form = new WWWForm();  form.AddField("name", nameField.text);  form.AddField("password", passwordField.text);  string url = "http://localhost:8888/3D\_Connect\_4/register.php";  using (UnityWebRequest www = UnityWebRequest.Post(url, form))  {  yield return www.SendWebRequest();  if (www.result != UnityWebRequest.Result.Success)  {  Debug.LogError("Network error:" + www.error);  yield break;  }  string reponseText = www.downloadHandler.text.Trim();  if (reponseText == "0")  {  Debug.Log("user created succcessfully"); UnityEngine.SceneManagement.SceneManager.LoadScene("GameUI");  }  else  {  errorMessage.text = "Username already exist";  }  }  }  public void VerifyInputs()  {  submitButton.interactable = (nameField.text.Length >= 8 && passwordField.text.Length >= 8);  }  } | |

### AI Mode

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598 | using System.Collections;  using System.Collections.Generic;  using System.ComponentModel.Design.Serialization;  using UnityEngine;  public class AI\_Mode : MonoBehaviour  {  public static AI\_Mode instance;  int numCols = 4;  int numRows = 4;  int numLayers = 4;  int maxSearch = DBManager.difficulty;  private long[,,,] zobristTable;  private Dictionary<long, float> transpositionTable = new Dictionary<long, float>();  private System.Random random = new System.Random();  public class Move  {  public int column;  public int row;  public int layer;  public float score;  public Move()  {  }  public Move(float \_score)  {  score = \_score;  }  public Move(int \_row, int \_column, int \_layer, float \_score)  {  row = \_row;  column = \_column;  layer = \_layer;  score = \_score;  }  public Move(int \_row, int \_column, int \_layer)  {  row = \_row;  column = \_column;  layer = \_layer;  }  }  private void Awake()  {  instance = this;  init\_zobrist();  Debug.Log(maxSearch);  }  private void init\_zobrist()  {  zobristTable = new long[numLayers, numRows, numCols, 2]; // Assuming 2 players  for (int layer = 0; layer < numLayers; layer++)  {  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  for (int player = 0; player < 2; player++) // 2 players  {  zobristTable[layer, row, col, player] = RandomLong();  }  }  }  }  }  private long RandomLong()  {  byte[] buffer = new byte[8];  random.NextBytes(buffer);  return System.BitConverter.ToInt64(buffer, 0);  }  // Assuming player values are 1 and 2, and empty is 0  private long ComputeHash(int[,,] board)  {  long hash = 0;  for (int layer = 0; layer < numLayers; layer++)  {  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {  int player = board[layer, row, col];  if (player > 0) // If not empty  {  hash ^= zobristTable[layer, row, col, player - 1];  }  }  }  }  return hash;  }  // Minimax, GetValidMoves, PerformTempMove, EvaluateBoard methods should be updated to use Zobrist hashing    List<Move> GetValidMoves(int[,,] currentBoard)  {  List<Move> moveList = new List<Move>();  for (int col = 0; col < numCols; col++)  {  for (int row = 0; row < numRows; row++)  {  for (int layer = numLayers - 1; layer >= 0; layer--)  {  if (currentBoard[layer, row, col] == 0)  {  Move m = new Move(row, col, layer);  moveList.Add(m);  break;  }  }  }  }  return moveList;  }  public void BestMove()//start the AI  {  Move bestMove = new Move(-1, -1, -1, Mathf.NegativeInfinity);  int[,,] currentPlayfield = Playfield.instance.CurrentPlayfield();  List<Move> possibleMoves = new List<Move>();  possibleMoves.AddRange(GetValidMoves(currentPlayfield));    foreach (Move move in possibleMoves)  {  move.score = Mathf.NegativeInfinity;  int[,,] tempBoard = PerformTempMove(move, currentPlayfield, 2);  move.score = Minimax(tempBoard, maxSearch, false, Mathf.NegativeInfinity, Mathf.Infinity);  if (move.score > bestMove.score)  {  bestMove = move;  }  }  //Game Manager - perform the best move  GameManager.instance.ColumnPressed(bestMove.row, bestMove.column); //switch the row and column if it doesn't work  }  int[,,] PerformTempMove(Move move, int[,,] currenBoard, int player)  {  int[,,] tempBoard = new int[numLayers, numRows, numCols];  System.Array.Copy(currenBoard, tempBoard, currenBoard.Length);  tempBoard[move.layer, move.row, move.column] = player;  return tempBoard;  }  float Minimax(int[,,] currentBoard, int depth, bool isMaximizer, float alpha, float beta)  {  long boardHash = ComputeHash(currentBoard);  if (transpositionTable.ContainsKey(boardHash))  {  return transpositionTable[boardHash];  }  if (depth == 0)  {  float score = EvaluateBoard(currentBoard, isMaximizer);  return score;    }  List<Move> possibleMoves = GetValidMoves(currentBoard);  if (isMaximizer)  {  float maxEval = Mathf.NegativeInfinity;  foreach (Move move in possibleMoves)  {  int[,,] tempBoard = PerformTempMove(move, currentBoard, 2); // Assuming 2 is AI player  float eval = Minimax(tempBoard, depth - 1, false, alpha, beta);  maxEval = Mathf.Max(maxEval, eval);  alpha = Mathf.Max(alpha, eval);  if (beta <= alpha)  break;  }  transpositionTable[boardHash] = maxEval;  return maxEval;  }  else  {  float minEval = Mathf.Infinity;  foreach (Move move in possibleMoves)  {  int[,,] tempBoard = PerformTempMove(move, currentBoard, 1); // Assuming 1 is opponent  float eval = Minimax(tempBoard, depth - 1, true, alpha, beta);  minEval = Mathf.Min(minEval, eval);  beta = Mathf.Min(beta, eval);  if (beta <= alpha)  break;  }  transpositionTable[boardHash] = minEval;  return minEval;  }  }  float EvaluateBoard(int[,,] currentBoard, bool isMaximizer)  {  float boardScore = 0;  boardScore += z\_axis\_check(currentBoard, isMaximizer);  boardScore += x\_axis\_check(currentBoard, isMaximizer);  boardScore += x\_z\_axis\_check(currentBoard, isMaximizer);  boardScore += y\_axis\_check(currentBoard, isMaximizer);  boardScore += y\_z\_axis\_check(currentBoard, isMaximizer);  boardScore += y\_x\_axis\_check(currentBoard, isMaximizer);  boardScore += x\_y\_z\_axis\_check(currentBoard, isMaximizer);  return boardScore;  }    //2D Check  float z\_axis\_check(int[,,] currentBoard, bool isMaximizer) //Horizontal Check  {  float score = 0;  for (int layer = numLayers - 1; layer >= 0; layer--)  {  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {    //Left to right  if (col < 1)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row, col + 1];  int c = currentBoard[layer, row, col + 2];  int d = currentBoard[layer, row, col + 3];  //WinCheck  score += WinCheck(a, b, c, d, isMaximizer);  }  //Right to Left  if (col > 2)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row, col - 1];  int c = currentBoard[layer, row, col - 2];  int d = currentBoard[layer, row, col - 3];  score += WinCheck(a, b, c, d, isMaximizer);  }  }    }  }  return score;  }  float x\_axis\_check(int[,,] currentBoard, bool isMaximizer)//Vertical Check  {  float score = 0;  for (int layer = numLayers - 1; layer >= 0; layer--)  {  for (int col = 0; col < numCols; col++)  {  for (int row = 0; row < numRows; row++)  {    //Left to right  if (row < 1)  {    int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row + 1, col];  int c = currentBoard[layer, row + 2, col];  int d = currentBoard[layer, row + 3 , col];  //WinCheck  score += WinCheck(a, b, c, d, isMaximizer);  }  //Right to Left  if (row > 2)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row - 1, col];  int c = currentBoard[layer, row - 2, col];  int d = currentBoard[layer, row - 3, col];  score += WinCheck(a, b, c, d, isMaximizer);  }  }  }  }    return score;  }  float x\_z\_axis\_check(int[,,] currentBoard, bool isMaximizer) //Diagonal Check  {  float score = 0;  //Top to bottom  for (int layer = numLayers - 1; layer >= 0; layer--)  {  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {    //right to left  if (row < 1 && col < 1)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row + 1, col + 1];  int c = currentBoard[layer, row + 2, col + 2];  int d = currentBoard[layer, row + 3, col + 3];  score += WinCheck(a, b, c, d, isMaximizer);  }  //left to right  if (row < 1 && col > 2)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row + 1, col - 1];  int c = currentBoard[layer, row + 2, col - 2];  int d = currentBoard[layer, row + 3, col - 3];  score += WinCheck(a, b, c, d, isMaximizer);  }    }  }  }  //Bottom to top  for (int layer = numLayers - 1; layer >= 0; layer--)  {  for (int row = numRows - 1; row >= 0; row--)  {  for (int col = 0; col < numCols; col++)  {    if (row > 2 && col < 1)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row - 1, col + 1];  int c = currentBoard[layer, row - 2, col + 2];  int d = currentBoard[layer, row - 3, col + 3];  score += WinCheck(a, b, c, d, isMaximizer);  }  //  if (row > 2 && col > 2)  {  int a = currentBoard[layer, row, col];  int b = currentBoard[layer, row - 1, col - 1];  int c = currentBoard[layer, row - 2, col - 2];  int d = currentBoard[layer, row - 3, col - 3];  score += WinCheck(a, b, c, d, isMaximizer);  }  }  }  }  return score;    }  //3D check  float y\_axis\_check(int[,,] currentBoard, bool isMaximizer)  {  float score = 0;  for (int row = 0; row < numRows; row++)  {  for (int col = 0; col < numCols; col++)  {    int a = currentBoard[3, row, col];  int b = currentBoard[2, row, col];  int c = currentBoard[1, row, col];  int d = currentBoard[0, row, col];  score += WinCheck(a, b, c, d, isMaximizer);  }  }    return score;  }  float y\_z\_axis\_check(int[,,] currentBoard, bool isMaximizer)  {  float score = 0;  for (int row = 0; row < numRows; row++)  {  //left to right    //Bottom to Top  int a = currentBoard[3, row, 0];  int b = currentBoard[2, row, 1];  int c = currentBoard[1, row, 2];  int d = currentBoard[0, row, 3];  score += WinCheck(a, b, c, d, isMaximizer);  //Top to bottom  int e = currentBoard[0, row, 0];  int f = currentBoard[1, row, 1];  int g = currentBoard[2, row, 2];  int h = currentBoard[3, row, 3];  score += WinCheck(e, f, g, h, isMaximizer);  //right to left    //Bottom to Top  int i = currentBoard[3, row, 3];  int j = currentBoard[2, row, 2];  int k = currentBoard[1, row, 1];  int l = currentBoard[0, row, 0];  score += WinCheck(i, j, k, l, isMaximizer);  //Top to Bottom  int m = currentBoard[0, row, 3];  int n = currentBoard[1, row, 2];  int o = currentBoard[2, row, 1];  int p = currentBoard[3, row, 0];  score += WinCheck(m, n, o, p, isMaximizer);      }  return score;  }  float y\_x\_axis\_check(int[,,] currentBoard, bool isMaximizer)  {  float score = 0;    for (int col = 0; col < numCols; col++)  {    //Front to back  //Bottom to Top  int a = currentBoard[3, 0, col];  int b = currentBoard[2, 1, col];  int c = currentBoard[1, 2, col];  int d = currentBoard[0, 3, col];  score += WinCheck(a, b, c, d, isMaximizer);  //Top to Bottom  int e = currentBoard[0, 0, col];  int f = currentBoard[1, 1, col];  int g = currentBoard[2, 2, col];  int h = currentBoard[3, 3, col];  score += WinCheck(e, f, g, h, isMaximizer);    //back to front  //Bottom to Top  int i = currentBoard[3, 3, col];  int j = currentBoard[2, 2, col];  int k = currentBoard[1, 1, col];  int l = currentBoard[0, 0, col];  score += WinCheck(i, j, k, l, isMaximizer);  int m = currentBoard[0, 3, col];  int n = currentBoard[1, 2, col];  int o = currentBoard[2, 1, col];  int p = currentBoard[3, 0, col];  score += WinCheck(m, n, o, p, isMaximizer);    }    return score;  }  float x\_y\_z\_axis\_check(int[,,] currentBoard, bool isMaximizer)  {  float score = 0;  //Bottom to Top  // top left to bottom right  int a = currentBoard[3, 0, 0];  int b = currentBoard[2, 1, 1];  int c = currentBoard[1, 2, 2];  int d = currentBoard[0, 3, 3];  score += WinCheck(a, b, c, d, isMaximizer);  //top right to bottom left  int e = currentBoard[3, 0, 3];  int f = currentBoard[2, 1, 2];  int g = currentBoard[1, 2, 1];  int h = currentBoard[0, 3, 0];  score += WinCheck(e, f, g, h, isMaximizer);  //bottom left to right  int i = currentBoard[3, 3, 0];  int j = currentBoard[2, 2, 1];  int k = currentBoard[1, 1, 2];  int l = currentBoard[0, 0, 3];  score += WinCheck(i, j, k, l, isMaximizer);  //bottom right to top left  int m = currentBoard[3, 3, 3];  int n = currentBoard[2, 2, 2];  int o = currentBoard[1, 1, 1];  int p = currentBoard[0, 0, 0];  score += WinCheck(m, n, o, p, isMaximizer);  //Top to bottom  //Top left to bottom right  int q = currentBoard[0, 0, 0];  int r = currentBoard[1, 1, 1];  int s = currentBoard[2, 2, 2];  int t = currentBoard[3, 3, 3];  score += WinCheck(q, r, s, t, isMaximizer);  //top right to bottom left  int u = currentBoard[0, 0, 3];  int v = currentBoard[1, 1, 2];  int w = currentBoard[2, 2, 1];  int x = currentBoard[3, 3, 0];  score += WinCheck(u, v, w, x, isMaximizer);  //bottom left to right  int A = currentBoard[0, 3, 0];  int B = currentBoard[1, 2, 1];  int C = currentBoard[2, 1, 2];  int D = currentBoard[3, 0, 3];  score += WinCheck(A, B, C, D, isMaximizer);  //bottom right to top left  int E = currentBoard[0, 3, 3];  int F = currentBoard[1, 2, 2];  int G = currentBoard[2, 1, 1];  int H = currentBoard[3, 0, 0];  score += WinCheck(E, F, G, H, isMaximizer);  return score;  }  private float WinCheck(int a, int b, int c, int d, bool isMaximizer)  {  float score = 0;  if ((a + b + c + d) == 0)  {  return 0;  }  if (a == b && a == c && a == d)  {  if (isMaximizer)  {  score += (a == 1) ? -1000 : 1000;  }  else //Minimizer  {  score += (a == 2) ? 1000 : -1000;  }  }  //3 Streak - But can be a four  if (a == b && a == c && d == 0)  {  if (isMaximizer)  {  score += (a == 1) ? -5 : 5;  }  else //Minimizer  {  score += (a == 2) ? 5 : -5;  }  }    //2 Streak- But can be a four  if (a == b && c == 0 && d == 0)  {  if (isMaximizer)  {  score += (a == 1) ? -1 : 1;  }  else //Minimizer  {  score += (a == 2) ? 1 : -1;  }  }  return score;  }  } | |

### Camera Rotation

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using TMPro;  using UnityEngine.UI;  using System.Xml.Serialization;  public class CameraRoatation : MonoBehaviour  {  [SerializeField] private Camera cam;  [SerializeField] private Transform target;  [SerializeField] private Button Up;  [SerializeField] private Button Down;  [SerializeField] private Button Left;  [SerializeField] private Button Right;  // Buttons for camera movement  private Vector3 previousPosition;  // Update is called once per frame  private float verticalAngle = 0.0f; // Track the vertical rotation angle  private float maxVerticalAngle = 90.0f; // Maximum angle to rotate upwards  private float minVerticalAngle = 25f; // Minimum angle to rotate downwards, allowing to go underneath    void Update()  {  if (Input.GetMouseButtonDown(0))  {  previousPosition = cam.ScreenToViewportPoint(Input.mousePosition);  }  if (Input.GetMouseButton(0))  {  //previousPositionds = cam.ScreenToViewportPoint(Input. mousePosition);  Vector3 direction = previousPosition - cam.ScreenToViewportPoint(Input.mousePosition);  Vector3 cameraRotation = cam.transform.rotation.eulerAngles;  // Calculate the new vertical angle, ensuring it stays within the desired range  verticalAngle += direction.y \* 180;  verticalAngle = Mathf.Clamp(verticalAngle, minVerticalAngle, maxVerticalAngle);  // Rotate the camera around the target horizontally  cam.transform.position = target.position;  cam.transform.Rotate(new Vector3(0, 1, 0), -direction.x \* 180, Space.World);    // Apply the clamped vertical rotation  cam.transform.rotation = Quaternion.Euler(verticalAngle, cam.transform.rotation.eulerAngles.y, 0);  // Move the camera back to maintain distance from the target  cam.transform.Translate(new Vector3(0, 0, -15));  previousPosition = cam.ScreenToViewportPoint(Input.mousePosition);  }  }  } | |

### Column Input

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class ColumnInput : MonoBehaviour  {  [SerializeField] private int row;  [SerializeField] private int column;    void OnMouseOver()  {  if (Input.GetMouseButtonDown(0))  {  Debug.Log("Column Presses:" + row + "," + column);  GameManager.instance.ColumnPressed(row, column);  // Access the BoxCollider component  BoxCollider boxCollider = GetComponent<BoxCollider>();  // Increase the size of the collider on each click  Vector3 newSize = boxCollider.size;  // newSize.y += 4;  // boxCollider.size = newSize;  Vector3 newCenter = boxCollider.center;  // newCenter.y += 3;  // boxCollider.center = newCenter;  }    }  } | |

### Game Manager

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270 | using System;  using System.Collections;  using System.Collections.Generic;  using Unity.Networking.Transport;  using UnityEngine;  using UnityEngine.SceneManagement;  using UnityEngine.UIElements;  using TMPro;  using UnityEditor.VersionControl;  using Unity.PlasticSCM.Editor.WebApi;  //For Client, handling drop block and loading scene when win  public class GameManager : MonoBehaviour  {    public static GameManager instance;  private SceneManagerNavigation sceneManagerNavigation;  // List to track all instantiated block prefabs  private List<GameObject> instantiatedBlocks = new List<GameObject>();  //bool activeTurn = true;  [SerializeField] private GameObject b\_block;  [SerializeField] private GameObject w\_block;  [SerializeField] private Transform startPoint;  [SerializeField] private GameObject gameOverWindow;  [SerializeField] private GameObject userNameDisplay;  [SerializeField] private GameObject playerTurnWindow;  [SerializeField] private TMP\_Text blackPlayerName;  [SerializeField] private TMP\_Text blackPlayerScore;  [SerializeField] private TMP\_Text whitePlayerName;  [SerializeField] private TMP\_Text whitePlayerScore;  [SerializeField] private TMP\_Text winner;  public TMP\_Text PlayerTurn;  public TMP\_Text win;  public Transform rematchIndicator;    bool activeTurn = true;  int currentPlayer = 1;  void Awake()  {  instance = this;  }  void Start()  {  gameOverWindow.SetActive(false);  userNameDisplay.SetActive(false);  if (!DBManager.online)  {  PlayerTurn.text = $”Player {currentPlayer}'s Turn";  }    }  public void SetUserDisplay(string usernameBlack, string usernameWhite, int scoreBlack, int scoreWhite)  {  userNameDisplay.SetActive(true);  blackPlayerName.text = "Black: " + usernameBlack;  blackPlayerScore.text = "Score: " + scoreBlack.ToString();  whitePlayerName.text = "White: " + usernameWhite;  whitePlayerScore.text = "Score: " + scoreWhite.ToString();  }  public void ColumnPressed(int row, int column)  {  if (DBManager.online)  {  int currentTeam = FindObjectOfType<ClientNetworkHandler>().GetCurrentTeam();  Debug.Log(currentTeam);  int layer = Playfield.instance.ValidMove(row, column);  NetMakeMove mm = new NetMakeMove();  mm.lastMoveRow = row;  mm.lastMoveColumn = column;  mm.lastMoveLayer = layer;  mm.teamId = currentTeam;  Client.Instance.SendToServer(mm);  }  else  {  if (!activeTurn)  {  Debug.Log("wait until turn is over");  return;  }  int layer = Playfield.instance.ValidMove(row, column);  if(layer != -1)  {  PlayBlock(row, column, layer, currentPlayer);  }  }  }    public void PlayBlock(int row, int column, int layer, int currentPlayer)  {  StartCoroutine(PlayBlockCourtine(row, column, layer, currentPlayer));  }    IEnumerator PlayBlockCourtine(int row, int column, int layer, int currentPlayer)  {  activeTurn = false;  GameObject block = (Instantiate(currentPlayer == 1 ? b\_block : w\_block)) as GameObject;    instantiatedBlocks.Add(block); // Track the instantiated block    block.transform.position = new Vector3(startPoint.position.x + column \* 2, startPoint.position.y, startPoint.position.z - row \* 2);    Vector3 goalPos = new Vector3(startPoint.position.x + column \* 2, startPoint.position.y - layer, startPoint.position.z - row \* 2);  while (MoveToGoal(goalPos, block)) { yield return null; }    Playfield.instance.DropBlock(layer, row, column, currentPlayer);  }  bool MoveToGoal(Vector3 goalPos, GameObject block)  {  return goalPos != (block.transform.position = Vector3.MoveTowards(block.transform.position, goalPos, 10f \* Time.deltaTime));  }    public void WinCondition(bool winner)  {  if (winner)  {  gameOverWindow.SetActive(true);  userNameDisplay.SetActive(false);  playerTurnWindow.SetActive(false);  rematchIndicator.gameObject.SetActive(true);  rematchIndicator.GetChild(0).gameObject.SetActive(false);  rematchIndicator.GetChild(1).gameObject.SetActive(false);  if (!DBManager.online)  {  win.text = (currentPlayer == 1 ? "Black Won" : "White Won");  }  }  else if(!DBManager.online)  {  activeTurn = true;  SwitchPlayer();  }  }  #region Options after game finshes  public void OnRematchButton()  {  if(DBManager.online)  {  NetRematch rm = new NetRematch();  rm.teamId = FindObjectOfType<ClientNetworkHandler>().GetCurrentTeam();  rm.wantRematch = 1; // 1 means want to rematch  Client.Instance.SendToServer(rm);  }  else  {  GameReset();  currentPlayer = 1;  PlayerTurn.text = (currentPlayer == 1 ? "Black's Turn" : "White's Turn");  Playfield.instance.Reset\_Board();  userNameDisplay.SetActive(false);  currentPlayer = 1;  activeTurn = true;  }    }  public void OnExitButton()  {  if(DBManager.online)  {  NetRematch rm = new NetRematch();  rm.teamId = FindObjectOfType<ClientNetworkHandler>().GetCurrentTeam();  rm.wantRematch = 0; // 0 means dont want to rematch  Client.Instance.SendToServer(rm);  SceneManager.LoadScene("GameUI");  }  else  {  SceneManager.LoadScene("GameUI");  }    }  public void OnNewGame()  {  if(!DBManager.online)  {  GameReset();  Playfield.instance.Reset\_Board();  userNameDisplay.SetActive(false);  currentPlayer = 1;  activeTurn = true;  }  if(DBManager.online)  {  NetRematch rm = new NetRematch();  rm.teamId = FindObjectOfType<ClientNetworkHandler>().GetCurrentTeam();  rm.wantRematch = 0; // 0 means dont want to rematch  Client.Instance.SendToServer(rm);  SceneManager.LoadScene("OnlineMenu");  }    }  #region restart the board  public void GameReset()  {  Reset\_Physical\_Board();  gameOverWindow.SetActive(false);  userNameDisplay.SetActive(true);  playerTurnWindow.SetActive(true);  }    private void Reset\_Physical\_Board()  {  //UI  rematchIndicator.transform.GetChild(0).gameObject.SetActive(false);  rematchIndicator.transform.GetChild(1).gameObject.SetActive(false);  // clean up  foreach (GameObject block in instantiatedBlocks)  {  Destroy(block);  }  instantiatedBlocks.Clear(); // Clear the list after destroying the objects  }    #endregion  #endregion  #region local game  void SwitchPlayer()  {  currentPlayer = (currentPlayer == 1 ? 2 : 1);  PlayerTurn.text = (currentPlayer == 1? "Player 1's Turn" : "Player 2's Turn");  if (currentPlayer == 2 && DBManager.ai)  {  AI\_Mode.instance.BestMove();  }    }  #endregion | |

### Outline Selection (Highlight function)

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.EventSystems;  public class OutlineSelection : MonoBehaviour  {  private Transform highlight;  private Transform selection;  private RaycastHit raycastHit;  void Update()  {  // Highlight  if (highlight != null)  {  highlight.gameObject.GetComponent<Outline>().enabled = false;  highlight = null;  }  Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);  if (!EventSystem.current.IsPointerOverGameObject() && Physics.Raycast(ray, out raycastHit))  {  highlight = raycastHit.transform;  if (highlight.CompareTag("Selectable") && highlight != selection)  {  Outline outline = highlight.gameObject.GetComponent<Outline>();  if (outline != null)  {  outline.enabled = true;  }  else  {  outline = highlight.gameObject.AddComponent<Outline>();  outline.enabled = true;  }  // Set the outline color to red and configure other properties  outline.OutlineColor = Color.red;  outline.OutlineWidth = 7.0f;  }  else  {  highlight = null;  }  }  // Selection  if (Input.GetMouseButtonDown(0) && highlight != null)  {  if (selection != null)  {  selection.gameObject.GetComponent<Outline>().enabled = false;  }  selection = highlight;  Outline selectionOutline = selection.gameObject.GetComponent<Outline>();  selectionOutline.enabled = true;  selectionOutline.OutlineColor = Color.red; // Ensure the selection outline is also red  highlight = null;  }  else if (Input.GetMouseButtonDown(0))  {  if (selection != null)  {  selection.gameObject.GetComponent<Outline>().enabled = false;  selection = null;  }  }  }  } | |

### Playfield

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438 | using System;  using System.Collections;  using System.Collections.Generic;  using TMPro;  using UnityEngine;  // For Clients  public class Playfield : MonoBehaviour  { // 0 = no coin, 1 = player1, 2 = player 2  public static Playfield instance;  const int rows = 4;  const int columns = 4;  const int layers = 4;  const int winlength = 4;  int[,,] board = new int[layers, rows, columns]  {  // Layer 0 (top)  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 1  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 2  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 3  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  }  };  private void Awake()  {  instance = this;  // DontDestroyOnLoad(gameObject);  }  public int ValidMove(int row, int column)  {  for (int layer = layers - 1 ; layer >= 0; layer--)  {  if (board[layer, row, column] == 0) //checking if (x,y) coordinate on that layer is free is free  {  return layer;  }  }  Debug.Log("No Valid Move");  return -1;  }    public void DropBlock(int layer, int row, int column, int player)  {  board[layer, row, column] = player;    print(DebugBoard());  if(!DBManager.online)  {  GameManager.instance.WinCondition(WinCheck());  }    }  #region LocalGame    bool WinCheck()  {  if (x\_axis\_Check() || z\_axis\_Check() || z\_x\_axis\_Check() || y\_axis\_Check() || x\_y\_axis\_Check() || z\_y\_axis\_Check() || x\_y\_z\_axis\_Check())  {  return true;    }  return false;  }  //2D Checks - each layer (start form the fourth to increase efficency)  bool x\_axis\_Check() //horizontal  {  for (int i = layers - 1; i >= 0; i--)  {  for (int j = 0; j < rows; j++)  {    if (board[i, j, 0] > 0)  {  int a = board[i, j, 0];  int b = board[i, j, 1];  int c = board[i, j, 2];  int d = board[i, j, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }    }  }  return false;  }  bool z\_axis\_Check() //Vertical  {  for (int i = layers - 1; i >= 0; i--)  {  for (int k = 0; k < columns; k++)  {    if (board[i, 0, k] > 0)  {  int a = board[i, 0, k];  int b = board[i, 1, k];  int c = board[i, 2, k];  int d = board[i, 3, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }    }  }  return false;  }  bool z\_x\_axis\_Check() //diagonal  {    for (int i = layers - 1; i >= 0; i--)  {  //Right to left Check  if (board[i, 0, 0] > 0)  {  int a = board[i, 0, 0];  int b = board[i, 1, 1];  int c = board[i, 2, 2];  int d = board[i, 3, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[i,0,3] > 0)  {  int a = board[i, 0, 3];  int b = board[i, 1, 2];  int c = board[i, 2, 1];  int d = board[i, 3, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }  return false;  }  //3D multi-layer involvement  bool y\_axis\_Check() //diagonal  {  for (int i = layers - 1; i >= 3; i--)  {  for (int j = 0; j < rows; j++)  {  for (int k = 0; k < columns; k++)  {  if (board[i, j, k] > 0)  {  int a = board[i, j, k];  int b = board[i - 1, j, k];  int c = board[i - 2, j, k];  int d = board[i - 3, j, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }  }  }  return false;  }  bool z\_y\_axis\_Check()  {    for (int j = 0; j < rows; j++)  {    if (board[3, j, 0] > 0)  {  int a = board[3, j, 0];  int b = board[2, j, 1];  int c = board[1, j, 2];  int d = board[0, j, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3, j, 3] > 0)  {  int a = board[3, j, 3];  int b = board[2, j, 2];  int c = board[1, j, 1];  int d = board[0, j, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }      }  return false;  }  bool x\_y\_axis\_Check()  {    for (int k = 0; k < columns; k++)  {  if (board[3, 0, k] > 0)  {  int a = board[3, 0, k];  int b = board[2, 1, k];  int c = board[1, 2, k];  int d = board[0, 3, k];    if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3, 3, k] > 0)  {  int a = board[3, 3, k];  int b = board[2, 2, k];  int c = board[1, 1, k];  int d = board[0, 0, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }  return false;  }  bool x\_y\_z\_axis\_Check() //through all layers  {  if (board[3,0,0] > 0)  {  int a = board[3, 0, 0];  int b = board[2, 1, 1];  int c = board[1, 2, 2];  int d = board[0, 3, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,0,3] > 0)  {  int a = board[3, 0, 3];  int b = board[2, 1, 2];  int c = board[1, 2, 1];  int d = board[0, 3, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,3,0] > 0)  {  int a = board[3, 3, 0];  int b = board[2, 2, 1];  int c = board[1, 1, 2];  int d = board[0, 0, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,3,3] > 0)  {  int a = board[3, 3, 3];  int b = board[2, 2, 2];  int c = board[1, 1, 1];  int d = board[0, 0, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  return false;  }  public int[,,] CurrentPlayfield()  {  int[,,] current = new int[layers, rows, columns];  System.Array.Copy(board, current, board.Length);  return current;  }  #endregion  string DebugBoard()  {  string s = "";  string seperator = ",";  string boarder = "|";  for (int i = 0; i < 4; i++) //layers  {  for (int j = 0; j < 4; j++) //rows  {  s += boarder;  for (int k = 0; k < 4; k++) //columns  {  s += board[i, j, k];  if (k != 3)  {  s += seperator;  }  else  {  s += boarder;  }  }  s += "\n";  }  s += "\n";  }  return s;  }  // resets the client UI board  public void Reset\_Board()  {  for (int i = 0; i < 4; i++) //layers  {  for (int j = 0; j < 4; j++) //rows  {  for (int k = 0; k < 4; k++) //columns  {  board[i,j,k] = 0;  }  }  }  }  } | |

### Server

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817 | using System;  using System.Collections;  using System.Collections.Generic;  using Unity.Collections;  using Unity.Networking.Transport;  using UnityEngine.Networking;  using UnityEngine;  using Newtonsoft.Json;  using System.Data;  public class Server : MonoBehaviour  {  public static Server Instance { set; get; }    private void Awake()  {  Instance = this;  }  public Dictionary<int, List<int>> rooms = new Dictionary<int, List<int>>(); //{Room: Player in room (InternalId)}  private Dictionary<int, GameRoom> gameRooms = new Dictionary<int, GameRoom>(); //{Room: type = Class GameRoom}  public Dictionary<int, UserInfo> userInfo = new Dictionary<int, UserInfo>(); //Internalid : Class UserInfo  private int roomCount = 1; //roomCount == SessionID //Initial room count //Retrive the largest session ID from the database  public NetworkDriver driver;  private NativeList<NetworkConnection> connections;    private bool isActive = false;  private const float keepAliveTickRate = 20.0f;  private float lastKeepAlive;  public Action connectionDropped;  public void Init(ushort port) // Start()  {  //init server  driver = NetworkDriver.Create(); //data are sent through this driver  NetworkEndPoint endpoint = NetworkEndPoint.AnyIpv4; //Who can connect to us  endpoint.Port = port;  if (driver.Bind(endpoint) != 0)  {  Debug.Log("unable to bind on port " + endpoint.Port);  return;  }  else  {  driver.Listen();  Debug.Log("Currently listening on port " + endpoint.Port);  }    //init the conenction list  connections = new NativeList<NetworkConnection>(8, Allocator.Persistent);//Max player in the server  isActive = true;  }  public void Shutdown()  {  if (isActive)  {  driver.Dispose();  connections.Dispose();  isActive = false;  }  }  public void OnDestroy()  {  Shutdown();  }  public void Update()  {  if (!isActive)  return;  KeepAlive();  driver.ScheduleUpdate().Complete();  CleanupConnections();  AcceptNewConnections();  UpdateMessagePump();  }  private void KeepAlive()  {  if (Time.time - lastKeepAlive > keepAliveTickRate)  {  lastKeepAlive = Time.time;  foreach (var room in rooms)  {  int roomNumber = room.Key;  List<int> playerIds = room.Value;  Broadcast(new NetKeepAlive(), playerIds);  }    }  }  private void CleanupConnections()  //Cleaning up stale connections ensures you don't have any old connections lying around when you iterate through the list to check for new events.  {  for (int i = 0; i < connections.Length; i++)  {  if (!connections[i].IsCreated)  {  connections.RemoveAtSwapBack(i);  --i;  }  }  }  private void AcceptNewConnections()  {  //Accept new connections  NetworkConnection c;  while ((c = driver.Accept()) != default(NetworkConnection))  {  connections.Add(c);  // AssignPlayerToRoom(c);  Debug.Log("Accept a connection");  }  }  public void AssignPlayerToRoom(NetworkConnection connection)  {  int playerId = connection.InternalId;  // Check if the player's score is available in userInfo  if (!userInfo.TryGetValue(playerId, out UserInfo joiningPlayerInfo))  {  Debug.LogError("Score not found for player: " + playerId);  return; // Or handle this case as needed  }  int joiningPlayerScore = joiningPlayerInfo.Score;  foreach (var room in rooms)  {  int roomNumber = room.Key;  List<int> roomPlayers = room.Value;  if (roomPlayers.Count == 1)  {  // Get the score of the existing player in the room  int existingPlayerId = roomPlayers[0];  if (userInfo.TryGetValue(existingPlayerId, out UserInfo existingPlayerInfo))  {  int scoreDifference = Math.Abs(existingPlayerInfo.Score - joiningPlayerScore);  if (scoreDifference <= 100)  {  // Retrieve the list of player IDs for this room  List<int> playersInRoom = rooms[roomNumber];  //Create a new GameRoom with the list of player IDs  gameRooms[roomNumber] = new GameRoom(playersInRoom);  Debug.Log("Game can start in room " + roomNumber);  // Add player to room  roomPlayers.Add(playerId);  Debug.Log($"Player {playerId} joined room {roomNumber}");    roomCount++;  return;  }  }  }  }  // If no suitable room is found, create a new one  rooms[roomCount] = new List<int> { playerId };  Debug.Log($"Player {playerId} created a new room {roomCount}");  }  public class UserInfo  {  public string Username { get; set; }  public int Score { get; set; }  public int userid { get; set; }  }  public void UpdateUserInfo(int id, string username, int score, int userid)  {  if (userInfo.ContainsKey(id))  {  userInfo[id].Username = username;  userInfo[id].Score = score;  userInfo[id].userid = userid;  }  else  {  userInfo[id] = new UserInfo { Username = username, Score = score };  print("sucess");  }  }  private void UpdateMessagePump()  {  DataStreamReader stream; //use to process received Data events  for (int i = 0; i < connections.Length; i++)  {  NetworkEvent.Type cmd;  while ((cmd = driver.PopEventForConnection(connections[i], out stream)) != NetworkEvent.Type.Empty)  {  if (cmd == NetworkEvent.Type.Data)  {  NetUtility.OnData(stream, connections[i], this);  }  else if (cmd == NetworkEvent.Type.Disconnect)  {  Debug.Log("Some client disconnected from server");  int playerId = connections[i].InternalId;  // Check if the player was assigned to a room  foreach (var room in rooms)  {  int roomNumber = room.Key;  List<int> roomPlayers = room.Value;  if (roomPlayers.Contains(playerId))  {  Debug.Log($"Player {playerId} disconnected from Room {roomNumber}");  roomPlayers.Remove(playerId); // Remove the player from the room  break;  }  }  connections[i] = default(NetworkConnection);  connectionDropped?.Invoke();  // Shutdown(); //Because we are in a 2player game, a shutdown is needed so if one of the player exits then the game will stop.  }  }  }  }  public void SendToClient(NetworkConnection connection, NetMessage msg)  {  DataStreamWriter writer;  driver.BeginSend(connection, out writer);  msg.Serialize(ref writer);  driver.EndSend(writer);  }  public void Broadcast(NetMessage msg, List<int> playerIds)  {  if (msg == null)  {  Debug.LogWarning("trying to boardcast a null message");  return;  }  Debug.Log($"Number of connections: {connections.Length}");  for (int i = 0; i < connections.Length; i++)  if (connections[i].IsCreated && playerIds.Contains(connections[i].InternalId))//sending to all clients in the room even the player who send the message  {  Debug.Log($"Sending {msg.Code} to :{connections[i].InternalId}");  SendToClient(connections[i], msg);    }  }  public GameRoom GetGameRoom(int roomId) {  if (gameRooms.TryGetValue(roomId, out GameRoom room)) {  return room;  }  return null; // Or handle this appropriately  }  public void StartStoringMovesCoroutine(List<int> connections, string jsonData, int winner)  {  StartCoroutine(StoreMove(connections, jsonData, winner));  }  IEnumerator StoreMove(List<int> connections, string json, int winner)  {  int playerConnectionId1 = connections[0];  int playerConnectionId2 = connections[1];  UserInfo player1Info = userInfo[playerConnectionId1];  UserInfo player2Info = userInfo[playerConnectionId2];  WWWForm form = new WWWForm();  form.AddField("p1\_id", player1Info.userid);  form.AddField("p2\_id", player2Info.userid);  form.AddField("winner\_id", winner == 1 ? player1Info.userid : player2Info.userid);  form.AddField("moves", json);  Debug.Log(winner == 1 ? player1Info.userid : player2Info.userid);  // Add the players name into the php script  string url = "http://localhost:8888/3D\_Connect\_4/savedata.php";  using (UnityWebRequest www = UnityWebRequest.Post(url, form))  {  yield return www.SendWebRequest();  if (www.result != UnityWebRequest.Result.Success)  {  Debug.LogError("Network error:" + www.error);  yield break;  }  string reponseText = www.downloadHandler.text.Trim();  if (reponseText == "0")  {  Debug.Log("Moves stored succcessfully");  }  else  {  Debug.Log("User creation failed. Error #" + www.downloadHandler.text);  }  }    }      public class GameRoom  {  const int rows = 4;  const int columns = 4;  const int layers = 4;  const int winlength = 4;  private List<Vector3Int> coordinateList = new List<Vector3Int>(); //<-- put into the database //[row layer column]  public int currentPlayer = 1;  public List<int> connectionID; //list of the conenctionid within the  public GameRoom(List<int> playerIDs) // Constructor to set the roomID  {  connectionID = playerIDs;  }  int[,,] board = new int[layers, rows, columns]  {  // Layer 0 (top)  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 1  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 2  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0}  },  // Layer 3  {  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  {0, 0, 0, 0},  }  };  #region WinCheck - return Bool    bool WinCheck()  {  if (x\_axis\_Check() || z\_axis\_Check() || z\_x\_axis\_Check() || y\_axis\_Check() || x\_y\_axis\_Check() || z\_y\_axis\_Check() || x\_y\_z\_axis\_Check())  {  return true;    }  return false;  }  #region Checks  //2D Checks - each layer (start form the fourth to increase efficency)  bool x\_axis\_Check() //horizontal  {  for (int i = layers - 1; i >= 0; i--)  {  for (int j = 0; j < rows; j++)  {    if (board[i, j, 0] > 0)  {  int a = board[i, j, 0];  int b = board[i, j, 1];  int c = board[i, j, 2];  int d = board[i, j, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }    }  }  return false;  }  bool z\_axis\_Check() //Vertical  {  for (int i = layers - 1; i >= 0; i--)  {  for (int k = 0; k < columns; k++)  {    if (board[i, 0, k] > 0)  {  int a = board[i, 0, k];  int b = board[i, 1, k];  int c = board[i, 2, k];  int d = board[i, 3, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }    }  }  return false;  }  bool z\_x\_axis\_Check() //diagonal  {    for (int i = layers - 1; i >= 0; i--)  {  //Right to left Check  if (board[i, 0, 0] > 0)  {  int a = board[i, 0, 0];  int b = board[i, 1, 1];  int c = board[i, 2, 2];  int d = board[i, 3, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[i,0,3] > 0)  {  int a = board[i, 0, 3];  int b = board[i, 1, 2];  int c = board[i, 2, 1];  int d = board[i, 3, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }  return false;  }  //3D multi-layer involvement  bool y\_axis\_Check() //diagonal  {  for (int i = layers - 1; i >= 3; i--)  {  for (int j = 0; j < rows; j++)  {  for (int k = 0; k < columns; k++)  {  if (board[i, j, k] > 0)  {  int a = board[i, j, k];  int b = board[i - 1, j, k];  int c = board[i - 2, j, k];  int d = board[i - 3, j, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }  }  }  return false;  }  bool x\_y\_axis\_Check()  {    for (int j = 0; j < rows; j++)  {    if (board[3, j, 0] > 0)  {  int a = board[3, j, 0];  int b = board[2, j, 1];  int c = board[1, j, 2];  int d = board[0, j, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3, j, 3] > 0)  {  int a = board[3, j, 3];  int b = board[2, j, 2];  int c = board[1, j, 1];  int d = board[0, j, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }      }  return false;  }  bool z\_y\_axis\_Check()  {    for (int k = 0; k < columns; k++)  {  if (board[3, 0, k] > 0)  {  int a = board[3, 0, k];  int b = board[2, 1, k];  int c = board[1, 2, k];  int d = board[0, 3, k];    if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3, 3, k] > 0)  {  int a = board[3, 3, k];  int b = board[2, 2, k];  int c = board[1, 1, k];  int d = board[0, 0, k];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  }      return false;  }  bool x\_y\_z\_axis\_Check() //through all layers  {  if (board[3,0,0] > 0)  {  int a = board[3, 0, 0];  int b = board[2, 1, 1];  int c = board[1, 2, 2];  int d = board[0, 3, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,0,3] > 0)  {  int a = board[3, 0, 3];  int b = board[2, 1, 2];  int c = board[1, 2, 1];  int d = board[0, 3, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,3,0] > 0)  {  int a = board[3, 3, 0];  int b = board[2, 2, 1];  int c = board[1, 1, 2];  int d = board[0, 0, 3];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  if (board[3,3,3] > 0)  {  int a = board[3, 3, 3];  int b = board[2, 2, 2];  int c = board[1, 1, 1];  int d = board[0, 0, 0];  if (a == b && a == c && a == d)  {  Debug.Log("Win" + a);  return true;  }  }  return false;  }  #endregion  #endregion  #region Game  public void DropBlock(int layer, int row, int column, int player)  {  board[layer, row, column] = player;    coordinateList.Add(new Vector3Int(row, column, layer));  //win check  WinCondition(WinCheck());  }  void SwitchPlayer()  {  currentPlayer = (currentPlayer == 1 ? 2 : 1);  Debug.Log($"Switched to Player {currentPlayer}");  }    #endregion  #region GameOver  public void WinCondition(bool winner)  {  if (winner)  {  //save the board  OnWinnerDetermined();  Reset\_Board();  //BroadcastMessage that there is a winner    NetEndGame endGameMsg = new NetEndGame();  endGameMsg.teamId = currentPlayer;  endGameMsg.EndGame = 1;  currentPlayer = 0;  Debug.Log("WinCondition");  Debug.Log($"Broadcasting end game message to players in room: {String.Join(", ", connectionID)}");  Server.Instance.Broadcast(endGameMsg, connectionID);  }  else  {  SwitchPlayer();  }    }  public void OnWinnerDetermined()  {  string jsonData = SerialiseJson(coordinateList);  Server.Instance.StartStoringMovesCoroutine(connectionID, jsonData, currentPlayer);  }  private string SerialiseJson(List<Vector3Int> vectorList)  {  // Convert the List<Vector3Int> into a List<List<int>>  var listOfLists = new List<List<int>>();  foreach (var vector in vectorList)  {  listOfLists.Add(new List<int> { vector.x, vector.y, vector.z });  }  // Serialize the List<List<int>> into a JSON string  string json = JsonConvert.SerializeObject(listOfLists);  return json;  }  public void Reset\_Board()  {  for (int i = 0; i < 4; i++) //layers  {  for (int j = 0; j < 4; j++) //rows  {  for (int k = 0; k < 4; k++) //columns  {  board[i,j,k] = 0;  }  }  }  }  #endregion  string DebugBoard()  {  string s = "";  string separator = ",";  string boarder = "|";  for (int i = 0; i < 4; i++) //layers  {  for (int j = 0; j < 4; j++) //rows  {  s += boarder;  for (int k = 0; k < 4; k++) //columns  {  s += board[i, j, k];  if (k != 3)  {  s += seperator;  }  else  {  s += boarder;  }  }  s += "\n";  }  s += "\n";  }  return s;  }  }  } | |

### Server Network Handler

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161 | using System;  using System.Collections;  using System.Collections.Generic;  using Newtonsoft.Json.Serialization;  using Unity.Networking.Transport;  using UnityEngine;  //Each person's networking stuff  public class ServerNetworkHandler : MonoBehaviour  {  private SceneManagerNavigation sceneManagerNavigation;  // private int playerCount = 0; // <--- For server  // Reference to the Server instance  private Server serverInstance;  private bool[] playerRematch = { false, false };  public void OnClickOnHost()  {  RegisterEvents();  DontDestroyOnLoad(gameObject);  }  private void RegisterEvents()  {  NetUtility.S\_WELCOME += OnWelcomeServer;  NetUtility.S\_MAKE\_MOVE += OnMakeMoveServer;  NetUtility.S\_REMATCH += OnRematchServer;  NetUtility.S\_END\_GAME += OnEndGameServer;  }  private void UnRegisterEvents()  {  NetUtility.S\_WELCOME -= OnWelcomeServer;  NetUtility.S\_MAKE\_MOVE -= OnMakeMoveServer;  NetUtility.S\_REMATCH -= OnRematchServer;  NetUtility.S\_END\_GAME -= OnEndGameServer;  }    //Server  private void OnWelcomeServer(NetMessage msg, NetworkConnection cnn)  {  //Client has connected, assign a team and return the message back to the client  NetWelcome nw = msg as NetWelcome;  int score = nw.score;  string username = nw.username;  int userid = nw.userid;  Server.Instance.UpdateUserInfo(cnn.InternalId, username, score, userid);  Server.Instance.AssignPlayerToRoom(cnn);  // //Assign a team ----------------------------------------------------------------  foreach (var room in Server.Instance.rooms)  {  if (room.Value.Contains(cnn.InternalId))  {  int playerIndex = room.Value.IndexOf(cnn.InternalId);  // Assign team based on player's position in the room  nw.AssignedTeam = playerIndex + 1;  Debug.Log($"player {cnn.InternalId} joined room {room.Key} with team number {nw.AssignedTeam}");  break;  }  }  //Store it into the dictionary  Server.Instance.SendToClient(cnn, nw);  // Server.Instance.UpdateUsername(cnn.InternalId, username );  //Return back to the client  foreach (var room in Server.Instance.rooms)  {  if (room.Value.Count == 2 && room.Value.Contains(cnn.InternalId))  {  NetStartGame nsg = new NetStartGame();  // Assuming the first player in the list is Black and the second is White  Server.UserInfo userInfoBlack = Server.Instance.userInfo[room.Value[0]];  Server.UserInfo userInfoWhite = Server.Instance.userInfo[room.Value[1]];  // Populate the NetStartGame message  nsg.scoreBlack = userInfoBlack.Score;  nsg.scoreWhite = userInfoWhite.Score;  nsg.usernameBlack = userInfoBlack.Username;  nsg.usernameWhite = userInfoWhite.Username;      // Call the BroadcastToRoom method in the Server script  Server.Instance.Broadcast(nsg, room.Value);  break;  }  }  }  private void OnMakeMoveServer(NetMessage msg, NetworkConnection cnn)  {  //Recieve the message, broadcast it back  NetMakeMove mm = msg as NetMakeMove;  //receive, and broadcast it back  foreach (var room in Server.Instance.rooms)  {  if (room.Value.Count == 2 && room.Value.Contains(cnn.InternalId))  {  //Make the move in the specific room in the GameRoom Class  Server.GameRoom Room = Server.Instance.GetGameRoom(room.Key);  if (mm.teamId == Room.currentPlayer)  {  // Call the BroadcastToRoom method in the Server script if the player who sent is the one who meant to make the move  Server.Instance.Broadcast(msg, room.Value);  Room.DropBlock(mm.lastMoveLayer,mm.lastMoveRow,mm.lastMoveColumn, mm.teamId);  }  break;  }  }  }  private void OnRematchServer(NetMessage msg, NetworkConnection cnn)  {  //receive, and broadcast it back  foreach (var room in Server.Instance.rooms)  {  if (room.Value.Count == 2 && room.Value.Contains(cnn.InternalId))  {  // Call the BroadcastToRoom method in the Server script  Server.Instance.Broadcast(msg, room.Value);  NetRematch rm = msg as NetRematch;  playerRematch[rm.teamId - 1] = rm.wantRematch == 1;  if (playerRematch[0] && playerRematch[1])  {  Server.GameRoom gameRoom = Server.Instance.GetGameRoom(room.Key);  gameRoom.currentPlayer = 1;  gameRoom.Reset\_Board();  }  break;  }      }    }    private void OnEndGameServer(NetMessage msg, NetworkConnection cnn)  {  }    } | |

### Client

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129 | using System;  using Unity.Collections;  using Unity.Networking.Transport;  using UnityEngine;  public class Client : MonoBehaviour  {  public static Client Instance { set; get; }  private void Awake()  {  Instance = this;  }  public NetworkDriver driver;  private NetworkConnection connection;  private bool isActive = false;  public Action connectionDropped;  public void Init(string ip, ushort port)  {  driver = NetworkDriver.Create();  NetworkEndPoint endpoint = NetworkEndPoint.Parse(ip, port);  endpoint.Port = port;  connection = driver.Connect(endpoint);  Debug.Log("Attempting to connect to Server on " + endpoint.Address);  isActive = true;  RegisterToEvent();  }  public void Shutdown()  {  if (isActive)  {  UnregisterToEvent();  driver.Dispose();  isActive = false;  connection = default(NetworkConnection);  }  }  public void OnDestroy()  {  Shutdown();  }  public void Update()  {  if (!isActive)  return;  driver.ScheduleUpdate().Complete();  CheckAlive();  UpdateMessagePump();  }  private void CheckAlive()  {  if (!connection.IsCreated && isActive)  {  Debug.Log("Lost connection to server");  connectionDropped?.Invoke();  Shutdown();  }  }  private void UpdateMessagePump() //Add the username into here  {  DataStreamReader stream;  NetworkEvent.Type cmd;  while ((cmd = connection.PopEvent(driver, out stream)) != NetworkEvent.Type.Empty)  {  if (cmd == NetworkEvent.Type.Connect)  {  NetWelcome welcomeMsg = new NetWelcome();  welcomeMsg.username = DBManager.username;  welcomeMsg.score = DBManager.score;  welcomeMsg.userid = DBManager.userid;  SendToServer(welcomeMsg);  // SendToServer()//Sending the server the Username  Debug.Log("We are Connect");  }  else if (cmd == NetworkEvent.Type.Data)  {  NetUtility.OnData(stream, default(NetworkConnection));  }  else if (cmd == NetworkEvent.Type.Disconnect)  {  Debug.Log("Client disconnected from server");  connectionDropped?.Invoke();  Shutdown();  }  }  }  public void SendToServer(NetMessage msg)  {  DataStreamWriter writer;  driver.BeginSend(connection, out writer);  msg.Serialize(ref writer);  driver.EndSend(writer);  }  // Event parsing  private void RegisterToEvent()  {  NetUtility.C\_KEEP\_ALIVE += OnKeepAlive;  }  private void UnregisterToEvent()  {  NetUtility.C\_KEEP\_ALIVE -= OnKeepAlive;  }  private void OnKeepAlive(NetMessage nm)  {  SendToServer(nm);  }  } | |

### Client Network Handler

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155 | using System;  using System.Collections;  using System.Collections.Generic;  using Unity.Networking.Transport;  using UnityEngine;  //Each person's networking stuff  public class ClientNetworkHandler : MonoBehaviour  {  public static ClientNetworkHandler Instance { set; get; }  private int currentTeam = 1; // <-- Different for each client  private SceneManagerNavigation sceneManagerNavigation;  private bool[] playerRematch = { false, false };  public void OnClickOnHost()  {  RegisterEvents();  DontDestroyOnLoad(gameObject);  Instance = this;  }  private void RegisterEvents()  {  NetUtility.C\_WELCOME += OnWelcomeClient;  NetUtility.C\_START\_GAME += OnStartGameClient;  NetUtility.C\_MAKE\_MOVE += OnMakeMoveClient;  NetUtility.C\_REMATCH += OnRematchClient;  NetUtility.C\_END\_GAME += OnEndGameClient;  }  private void UnRegisterEvents()  {  NetUtility.C\_WELCOME -= OnWelcomeClient;  NetUtility.C\_START\_GAME -= OnStartGameClient;  NetUtility.C\_REMATCH -= OnRematchClient;  NetUtility.C\_END\_GAME -= OnEndGameClient;  }  public void CallUnRegisterEvents()  {  UnRegisterEvents();  }  public int GetCurrentTeam() //used in game GameManager  {  return currentTeam;  }  private void OnWelcomeClient(NetMessage msg)  {  //recieve the connection message  NetWelcome nw = msg as NetWelcome;  //assign team  currentTeam = nw.AssignedTeam;  Debug.Log($"My assigned team is {nw.AssignedTeam}");  }  private void OnStartGameClient(NetMessage msg)  {    sceneManagerNavigation = FindObjectOfType<SceneManagerNavigation>();  sceneManagerNavigation.GoToOnlineGame();  StartCoroutine(SetupUI(msg as NetStartGame));    }  private IEnumerator SetupUI(NetStartGame msg)  {  // Wait until the new scene has started (you might need a frame delay)  yield return null; // Optional: yield return new WaitUntil(() => some condition);    if (GameManager.instance != null)  {  GameManager.instance.SetUserDisplay(msg.usernameBlack, msg.usernameWhite,msg.scoreBlack, msg.scoreWhite);  GameManager.instance.PlayerTurn.text = (currentTeam == 1? "Your Turn" : "Opponent's Turn");  DBManager.online = true;  }  else  {  Debug.LogError("GameManager instance is not set after scene load.");  }    }    private void OnMakeMoveClient(NetMessage msg)  {  NetMakeMove mm = msg as NetMakeMove;  Debug.Log($"MM : CurrentPlayer - {mm.teamId},Row - {mm.lastMoveRow}, Column - {mm.lastMoveColumn}, Layer - {mm.lastMoveLayer}");  GameManager.instance.PlayBlock(mm.lastMoveRow, mm.lastMoveColumn, mm.lastMoveLayer, mm.teamId);  if (GameManager.instance.PlayerTurn.text == "Your Turn")  {  GameManager.instance.PlayerTurn.text = "Opponent's Turn";  }  else  {  GameManager.instance.PlayerTurn.text = "Your Turn";  }  }      private void OnRematchClient(NetMessage msg)  {  NetRematch rm = msg as NetRematch;  playerRematch[rm.teamId - 1] = rm.wantRematch == 1; //Return Bool if == 1 -> true  //Activate the rematchIndicator  if (rm.teamId != currentTeam)  {  GameManager.instance.rematchIndicator.GetChild((rm.wantRematch == 1) ? 0 : 1).gameObject.SetActive(true);    //Set the rematchButton to be false  }    if (playerRematch[0] && playerRematch[1])  {  //ResetBoard physically  GameManager.instance.GameReset();  //Change who goes first  currentTeam = (currentTeam == 1 ? 2 : 1);  GameManager.instance.PlayerTurn.text = (currentTeam == 1 ? "Your Turn" : "Opponent's Turn");  //Reset the clients' board  Playfield.instance.Reset\_Board();  //Reset the server's board which is storing in the database  }  }  private void OnEndGameClient(NetMessage msg)  {  NetEndGame em = msg as NetEndGame;  if (em != null && em.EndGame == 1)  {  Debug.Log("Game ended. Winning team: " + em.teamId);  GameManager.instance.win.text = (em.teamId == 1 ? "Black Won" : "White Won");  GameManager.instance.WinCondition(true);  }    }  } | |

### Network Manager (Joining Games)

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | using TMPro;  using UnityEngine;  using UnityEngine.SceneManagement;  public class NetworkManager : MonoBehaviour  {  public static NetworkManager Instance { set; get; }  public Server server;  public Client client;  [SerializeField] private TMP\_InputField addressInput;    public bool backButtonClicked = false;  private void Start()  {  addressInput = GameObject.Find("InputField (TMP)").GetComponent<TMP\_InputField>();  }  private void Awake()  {  if (Instance != null && Instance != this)  {  Destroy(Instance.gameObject);  }  Instance = this;  DontDestroyOnLoad(gameObject);  }  public void OnOnlineHostButton()  {  server.Init(8007);  SceneManager.LoadScene("HostMenu");  }  public void OnOnlineConnectButton()  {  string ipAddress = addressInput.text; // Use the input field value as the IP address  client.Init(ipAddress, 8007);  SceneManager.LoadScene("HostMenu");  }  public void OnHostBackButton()  {  client.Shutdown();  Destroy(gameObject);  Instance = null;  }  public bool BackButtonClicked  {  get { return backButtonClicked; }  }  } | |

### NetUtility

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | using System;  using Unity.Networking.Transport;  using UnityEngine;  public enum OpCode  {  KEEP\_ALIVE = 1,  WELCOME = 2,  START\_GAME = 3,  MAKE\_MOVE = 4,  END\_GAME = 5,  REMATCH = 6  }  public static class NetUtility  {  public static void OnData(DataStreamReader stream, NetworkConnection cnn, Server server = null)  {  NetMessage msg = null;  var opCode = (OpCode)stream.ReadByte();  switch (opCode)  {  case OpCode.KEEP\_ALIVE: msg = new NetKeepAlive(stream); break;  case OpCode.WELCOME: msg = new NetWelcome(stream); break;  case OpCode.START\_GAME: msg = new NetStartGame(stream); break;  case OpCode.MAKE\_MOVE: msg = new NetMakeMove(stream); break;  case OpCode.END\_GAME: msg = new NetEndGame(stream); break;  case OpCode.REMATCH: msg = new NetRematch(stream); break;  default:  Debug.LogError($"Message recevied had no Opcode{opCode}");  break;  }    if (msg != null)  {  if (server != null)  {  msg.ReceivedOnServer(cnn);  }  else  {  msg.ReceivedOnClient();  }  }  }  //Client  public static Action<NetMessage> C\_KEEP\_ALIVE;  public static Action<NetMessage> C\_WELCOME;  public static Action<NetMessage> C\_START\_GAME;  public static Action<NetMessage> C\_MAKE\_MOVE;  public static Action<NetMessage> C\_END\_GAME;  public static Action<NetMessage> C\_REMATCH;    //Server  public static Action<NetMessage, NetworkConnection> S\_KEEP\_ALIVE;  public static Action<NetMessage, NetworkConnection> S\_WELCOME;  public static Action<NetMessage, NetworkConnection> S\_START\_GAME;  public static Action<NetMessage, NetworkConnection> S\_MAKE\_MOVE;  public static Action<NetMessage, NetworkConnection> S\_END\_GAME;  public static Action<NetMessage, NetworkConnection> S\_REMATCH;  } | |

### NetMessage

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | using Unity.Networking.Transport;  using UnityEngine;  public class NetMessage  {  public OpCode Code { set; get; }  public virtual void Serialize(ref DataStreamWriter writer) //put stuff inside  {  writer.WriteByte((byte)Code);  }    public virtual void Deserialize(DataStreamReader reader) //unpacking and putting, and making sure it suits  {  }    public virtual void ReceivedOnClient()  {  }    public virtual void ReceivedOnServer(NetworkConnection cnn)  {  }  } | |

#### NetEndgame:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | using UnityEngine;  using Unity.Networking.Transport;  public class NetEndGame : NetMessage  {  public int teamId;  public byte EndGame; //1 = winner  public NetEndGame()  {  Code = OpCode.END\_GAME;  }  public NetEndGame(DataStreamReader reader)  {  Code = OpCode.END\_GAME;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  writer.WriteInt(teamId); //should I do teamID or use their playerId? if it doesn't work consider that  writer.WriteByte(EndGame); //1 = positive  }  public override void Deserialize(DataStreamReader reader)  {  teamId = reader.ReadInt();  EndGame = reader.ReadByte();  }  public override void ReceivedOnClient()  {  NetUtility.C\_END\_GAME?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_END\_GAME?.Invoke(this, cnn);  }  } | |

#### NetKeepAlive

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | using Unity.Networking.Transport;  public class NetKeepAlive : NetMessage  {  public NetKeepAlive()// <-- Making the box  {  Code = OpCode.KEEP\_ALIVE;  }  public NetKeepAlive(DataStreamReader reader) //< -- Receiving the box  {  Code = OpCode.KEEP\_ALIVE;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  }  public override void Deserialize(DataStreamReader reader)  {  }  public override void ReceivedOnClient()  {  NetUtility.C\_KEEP\_ALIVE?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_KEEP\_ALIVE?.Invoke(this, cnn);  }  } | |

#### NetMakeMove

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | using UnityEngine;  using Unity.Networking.Transport;  public class NetMakeMove : NetMessage  {  public int lastMoveLayer;  public int lastMoveRow;  public int lastMoveColumn;  public int teamId;  public NetMakeMove()  {  Code = OpCode.MAKE\_MOVE;  }  public NetMakeMove(DataStreamReader reader)  {  Code = OpCode.MAKE\_MOVE ;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  writer.WriteInt(lastMoveRow);  writer.WriteInt(lastMoveColumn);  writer.WriteInt(lastMoveLayer);  writer.WriteInt(teamId);    }  public override void Deserialize(DataStreamReader reader)  {  lastMoveRow = reader.ReadInt();  lastMoveColumn = reader.ReadInt();  lastMoveLayer = reader.ReadInt();  teamId = reader.ReadInt();    }  public override void ReceivedOnClient()  {  NetUtility.C\_MAKE\_MOVE?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_MAKE\_MOVE?.Invoke(this, cnn);  }  } | |

#### NetRematch

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | using UnityEngine;  using Unity.Networking.Transport;  public class NetRematch : NetMessage  {  public int teamId;  public byte wantRematch; //would this client want a rematch  public NetRematch()  {  Code = OpCode.REMATCH;  }  public NetRematch(DataStreamReader reader)  {  Code = OpCode.REMATCH;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  writer.WriteInt(teamId); //should I do teamID or use their playerId? if it doesn't work consider that  writer.WriteByte(wantRematch); //1 = positive  }  public override void Deserialize(DataStreamReader reader)  {  teamId = reader.ReadInt();  wantRematch = reader.ReadByte();  }  public override void ReceivedOnClient()  {  NetUtility.C\_REMATCH?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_REMATCH?.Invoke(this, cnn);  }  } | |

#### NetStartGame

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92 | using System.Collections;  using System.Collections.Generic;  using Unity.Networking.Transport;  using UnityEngine;  using Unity.Collections;  public class NetStartGame : NetMessage  {  public string usernameBlack;  public int scoreBlack;  public string usernameWhite;  public int scoreWhite;    public NetStartGame()  {  Code = OpCode.START\_GAME;  }  public NetStartGame(DataStreamReader reader)  {  Code = OpCode.START\_GAME;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  writer.WriteInt(scoreBlack);  writer.WriteInt(scoreWhite);  // Serialize usernameBlack  byte[] usernameBlackBytes = System.Text.Encoding.UTF8.GetBytes(usernameBlack);  using (var nativeUsernameBlackBytes = new NativeArray<byte>(usernameBlackBytes, Allocator.Temp))  {  writer.WriteInt(nativeUsernameBlackBytes.Length); // Write the length of the usernameBlack  writer.WriteBytes(nativeUsernameBlackBytes); // Then write the usernameBlack itself  }  // Serialize usernameWhite  byte[] usernameWhiteBytes = System.Text.Encoding.UTF8.GetBytes(usernameWhite);  using (var nativeUsernameWhiteBytes = new NativeArray<byte>(usernameWhiteBytes, Allocator.Temp))  {  writer.WriteInt(nativeUsernameWhiteBytes.Length); // Write the length of the usernameWhite  writer.WriteBytes(nativeUsernameWhiteBytes); // Then write the usernameWhite itself  }    }  public override void Deserialize(DataStreamReader reader)  {  scoreBlack = reader.ReadInt();  scoreWhite = reader.ReadInt();  // Deserialize usernameBlack  int usernameBlackLength = reader.ReadInt();  using (NativeArray<byte> usernameBlackBytes = new NativeArray<byte>(usernameBlackLength, Allocator.Temp))  {  reader.ReadBytes(usernameBlackBytes);  usernameBlack = System.Text.Encoding.UTF8.GetString(usernameBlackBytes.ToArray());  }  // Deserialize usernameWhite  int usernameWhiteLength = reader.ReadInt();  using (NativeArray<byte> usernameWhiteBytes = new NativeArray<byte>(usernameWhiteLength, Allocator.Temp))  {  reader.ReadBytes(usernameWhiteBytes);  usernameWhite = System.Text.Encoding.UTF8.GetString(usernameWhiteBytes.ToArray());  }  }  public override void ReceivedOnClient()  {  NetUtility.C\_START\_GAME?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_START\_GAME?.Invoke(this, cnn);  }  } | |

#### NetWelcome

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | using UnityEngine;  using Unity.Networking.Transport;  using Unity.Collections;  public class NetWelcome : NetMessage  {  public int AssignedTeam { set; get; }  public string username;  public int score;  public int userid;  public NetWelcome()  {  Code = OpCode.WELCOME;  }  public NetWelcome(DataStreamReader reader)  {  Code = OpCode.WELCOME;  Deserialize(reader);  }  public override void Serialize(ref DataStreamWriter writer)  {  writer.WriteByte((byte)Code);  writer.WriteInt(AssignedTeam);  writer.WriteInt(score);  writer.WriteInt(userid);    // Convert byte[] to NativeArray<byte>  byte[] usernameBytes = System.Text.Encoding.UTF8.GetBytes(username);  using (var nativeUsernameBytes = new NativeArray<byte>(usernameBytes, Allocator.Temp))  {  writer.WriteInt(nativeUsernameBytes.Length); // Write the length of the string  writer.WriteBytes(nativeUsernameBytes); // Then write the string itself  }  }  public override void Deserialize(DataStreamReader reader)  {  AssignedTeam = reader.ReadInt();  score = reader.ReadInt();  userid = reader.ReadInt();  //Deserialize the string  int usernameLength = reader.ReadInt(); // Read the length of the string  using (NativeArray<byte> usernameBytes = new NativeArray<byte>(usernameLength, Allocator.Temp))  {  reader.ReadBytes(usernameBytes); // Read the bytes into the array  username = System.Text.Encoding.UTF8.GetString(usernameBytes.ToArray()); // Convert bytes to string  }  }  public override void ReceivedOnClient()  {  NetUtility.C\_WELCOME?.Invoke(this);  }  public override void ReceivedOnServer(NetworkConnection cnn)  {  NetUtility.S\_WELCOME?.Invoke(this, cnn);    }  } | |

### Game Entry

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Networking;  using TMPro;  using UnityEngine.SceneManagement;  using Unity.VisualScripting;  public class GameEntry : MonoBehaviour  {  public static GameEntry instance;  public GameObject gameEntryPrefab; // Assign this in the Inspector  public Transform contentPanel; // Assign the ScrollView content panel  [System.Serializable]  public class GameDataList  {  public List<GameData> gameList;  }  [System.Serializable]  public struct GameData  {  public string SessionID;  public string OpponentUsername;  public string WinnerUsername;  public string OpponentScore;  public string UserRole;  public int TotalMoves; // Use the appropriate type for moves  }  void Start()  {  StartCoroutine(FetchGameHistory(DBManager.userid));  }  void Awake()  {  instance = this;  DontDestroyOnLoad(gameObject);  }  IEnumerator FetchGameHistory(int userid)  {  WWWForm form = new WWWForm();  form.AddField("userid", userid);  string url = "http://localhost:8888/3D\_Connect\_4/retreivedata.php";  using (UnityWebRequest www = UnityWebRequest.Post(url, form))  {  yield return www.SendWebRequest();  if (www.result == UnityWebRequest.Result.Success)  {  Debug.Log(www.downloadHandler.text);  GameDataList gameDataList = JsonUtility.FromJson<GameDataList>("{\"gameList\":" + www.downloadHandler.text + "}");  PopulateGameHistoryUI(gameDataList.gameList);  }  else  {  Debug.LogError("Error in fetching game history: " + www.error);  }    }    }  private void PopulateGameHistoryUI(List<GameData> gameHistory)  {  foreach (var child in contentPanel.GetComponentsInChildren<GameEntry>())  {  Destroy(child.gameObject); // Clear existing entries  }  foreach (GameData data in gameHistory)  {  GameObject entryObject = Instantiate(gameEntryPrefab, contentPanel);  TMP\_Text[] texts = entryObject.GetComponentsInChildren<TMP\_Text>();  UnityEngine.UI.Button entryButton = entryObject.GetComponentInChildren<UnityEngine.UI.Button>();  texts[0].text = data.UserRole == "White" ? $"{DBManager.username} ({DBManager.score})" : $"{data.OpponentUsername} ({data.OpponentScore})";  texts[1].text = data.UserRole == "White" ? $"{data.OpponentUsername} ({data.OpponentScore})" : $"{DBManager.username} ({DBManager.score})" ;  texts[2].text = data.WinnerUsername == DBManager.username ? "Win" : "Lose";  texts[3].text = data.TotalMoves.ToString();  // Add an event listener to the button that passes the specific GameData instance  entryButton.onClick.AddListener(() => OnGameEntryClick(data));  }  }  private void OnGameEntryClick(GameData data)  {  // Now you have the data for the specific game entry that was clicked  // You can use this data to transition to the replay scene or do something else  SceneManager.LoadScene("Replay");  Debug.Log($"Game entry clicked with Session ID: {data.SessionID}");    // For example, if you want to start a replay:  StartCoroutine(CallReplay(data));    }  private IEnumerator CallReplay(GameData data)  {    yield return null;  Debug.Log("Replay");  Replay.instance.CallReplay(data);    }  } | |

#### Replay

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186 | using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.Networking;  using UnityEngine.UI;  using Newtonsoft.Json;  using TMPro;  using UnityEngine.UIElements;  public class Replay : MonoBehaviour  {  public static Replay instance;  [SerializeField] private GameObject b\_block;  [SerializeField] private GameObject w\_block;  [SerializeField] private Transform startPoint;  [SerializeField] private TMP\_Text P1\_User;  [SerializeField] private TMP\_Text P2\_User;  private List<GameObject> instantiatedBlocks = new List<GameObject>();  private List<Vector3Int> moves = new List<Vector3Int>(); //List to track moves [row, column, layer]  int currentPlayer = 1;  [SerializeField] private TMP\_Text Winner;  private bool isKeyPressed = false;    int pointer = 0; //which move through the list the client is on  void Awake()  {  instance = this;  }  public void CallReplay(GameEntry.GameData data)  {  StartCoroutine(GetMove(data)); //Edit this so clients can choose from  }  IEnumerator GetMove(GameEntry.GameData data)  {  WWWForm form = new WWWForm();  form.AddField("SessionID", data.SessionID);  string url = "http://localhost:8888/3D\_Connect\_4/replay.php";  using (UnityWebRequest www = UnityWebRequest.Post(url, form))  {  yield return www.SendWebRequest();  string responseText = www.downloadHandler.text;  string[] responseParts = responseText.Split('\t');  if (responseParts[0] == "0" )  {  //Data in reponseParts[1]  string movesJson = responseParts[1];  P1\_User.text = data.UserRole == "White" ? $"White: {DBManager.username} ({DBManager.score})" : $"White: {data.OpponentUsername} ({data.OpponentScore})";  P2\_User.text = data.UserRole == "White" ? $"Black: {data.OpponentUsername} ({data.OpponentScore})" : $"Black: {DBManager.username} ({DBManager.score})" ;  Winner.text = $"Winner : {data.WinnerUsername}";  DeserialiseJson(movesJson);  }  }  }  private void DeserialiseJson(string json)  {  // Deserialize the JSON string into a List<List<int>>  var listOfLists = JsonConvert.DeserializeObject<List<List<int>>>(json);  if (listOfLists != null)  {  foreach (var coord in listOfLists)  {  // Assuming each inner list contains exactly 3 integers (x, y, z)  if (coord.Count == 3)  {  moves.Add(new Vector3Int(coord[0], coord[1], coord[2]));    }  }  }  }  void Update()  {  // Detect right arrow key for forward  if (Input.GetKeyDown(KeyCode.RightArrow) && !isKeyPressed)  {  OnForwardArrow();  isKeyPressed = true;  }  else if (Input.GetKeyUp(KeyCode.RightArrow))  {  isKeyPressed = false;  }  // Detect left arrow key for backward  if (Input.GetKeyDown(KeyCode.LeftArrow) && !isKeyPressed)  {  OnBackArrow();  isKeyPressed = true;  }  else if (Input.GetKeyUp(KeyCode.LeftArrow))  {  isKeyPressed = false;  }  }  public void OnForwardArrow()  {  if (pointer < moves.Count)  {  var move = moves[pointer];  PlayBlock(move.x, move.y,move.z);  pointer++;  }  }  public void OnBackArrow()  {  if (pointer > 0)  {  pointer--;  var move = moves[pointer];  RemoveBlock(move.x, move.y, move.z);  }  }  #region Place Block  public void PlayBlock(int row, int column, int layer)  {    StartCoroutine(PlayBlockCourtine(row, column, layer));  }  IEnumerator PlayBlockCourtine(int row, int column, int layer)  {  GameObject block = (Instantiate(currentPlayer == 1 ? b\_block : w\_block)) as GameObject;  instantiatedBlocks.Add(block); // Track the instantiated block  block.transform.position = new Vector3(startPoint.position.x + column \* 2, startPoint.position.y, startPoint.position.z - row \* 2);  Vector3 goalPos = new Vector3(startPoint.position.x + column \* 2, startPoint.position.y - layer, startPoint.position.z - row \* 2);  while (MoveToGoal(goalPos, block)) { yield return null; }  SwitchPlayer();  }      #endregion    #region Remove Block  public void RemoveBlock(int row, int column, int layer)  {  StartCoroutine(RemoveBlockCoroutine(row, column, layer));  }  IEnumerator RemoveBlockCoroutine(int row, int coloumn, int layer)  {  var block = instantiatedBlocks[pointer];  Vector3 goalPos = block.transform.position + Vector3.up \* 2;//adjust the height as needed  while (MoveToGoal(goalPos, block)) {yield return null;}  Destroy(block);  instantiatedBlocks.RemoveAt(pointer);  SwitchPlayer();  }  bool MoveToGoal(Vector3 goalPos, GameObject block)  {  return goalPos != (block.transform.position = Vector3.MoveTowards(block.transform.position, goalPos, 10f \* Time.deltaTime));  }  #endregion  void SwitchPlayer()  {  currentPlayer = (currentPlayer == 1 ? 2 : 1);  Debug.Log($"Switched to Player {currentPlayer}");  } | |

### SceneManagerNavigation

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126 | using System;  using System.Collections;  using UnityEditor.VersionControl;  using UnityEngine;  using UnityEngine.SceneManagement;  public class SceneManagerNavigation : MonoBehaviour  {  public void GoToRegister()  {  SceneManager.LoadScene("RegisterMenu");  }  public void GoToLogin()  {  SceneManager.LoadScene("LoginMenu");  }  public void GoToMenu()  {  SceneManager.LoadScene("GameUI");  }  //public void GoToLocalMenu() //1) create an AI option and a local play against your friend option  //{  // //SceneManager - go to LocalMenu  //}  public void GoToOnlineMenu()  {  SceneManager.LoadScene("OnlineMenu");  }  public void GoToOnlineFromHost()  {  NetworkManager onlineInstance = FindObjectOfType<NetworkManager>();  // Check if the button has been clicked  if (onlineInstance != null)  {  onlineInstance.backButtonClicked = true;  if (onlineInstance.BackButtonClicked)  {  onlineInstance.OnHostBackButton();  // Set the flag to true to indicate that the back button has been handled  onlineInstance.backButtonClicked = false;  }  Destroy(onlineInstance.gameObject);  SceneManager.LoadScene("OnlineMenu");  }  }  public void ExitGame()  {  if (DBManager.online)  {  Client.Instance.Shutdown();  ClientNetworkHandler.Instance.CallUnRegisterEvents();  SceneManager.LoadScene("GameUI");  }  else  {  SceneManager.LoadScene("GameUI");  }  }  public void GoToReplayMenu()  {  SceneManager.LoadScene("ReplayMenu");  }  public void GoToReplay()  {  SceneManager.LoadScene("Replay");  }  public void GoToOnlineGame()  {  SceneManager.LoadScene("Game");  }  #region AI Difficulity  public void AI()  {  SceneManager.LoadScene("AIDifficulty");  }  public void EasyMode()  {  DBManager.ai = true;  DBManager.difficulty = 1;  SceneManager.LoadScene("Game");    }  public void MediumMode()  {  DBManager.ai = true;  DBManager.difficulty = 2;  SceneManager.LoadScene("Game");    }  public void HardMode()  {  DBManager.ai = true;  DBManager.difficulty = 5;  SceneManager.LoadScene("Game");  }  #endregion  public void GoToLocalGame()  {  SceneManager.LoadScene("Game");  }  public void LogOut()  {  DBManager.LogOut();  GoToMenu();  }  public void GoToLogOut()  {  SceneManager.LoadScene("LogOut");  }  } | |

PHP/ API Request:  
Registration API

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <?php  $con = mysqli\_connect('localhost', 'root', 'root', '3D\_Connect\_4');  //check that connection happened  if(mysqli\_connect\_errno()){  echo "1: connection failed"; // 1 = connection failed  exit();  }  $username = $\_POST["name"];  $password = $\_POST["password"];  //Check duplicates name  $namecheckquery = "SELECT username FROM UserID WHERE username = '" . $username . "';";  $namecheck = mysqli\_query($con, $namecheckquery) or die("2: name check query failed "); //2 = name check query failed  if (mysqli\_num\_rows($namecheck) > 0){  echo "3: name duplication"; //name duplication  exit();  }  //add user to table // check on the security do somemore research  $salt = "\$5\$rounds=5000\$"."bao".$username."\$";  $hash = crypt($password, $salt);  $insertuserquery = "INSERT INTO UserID (username, hash ,salt) VALUES ('".$username."','".$hash."','".$salt."');";  mysqli\_query($con, $insertuserquery) or die("4: Insert Player query failed"); //Insert Player query failed  echo("0");  ?> | |

#### Login API

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | <?php  $con = mysqli\_connect('localhost', 'root', 'root', '3D\_Connect\_4');  // Check that connection happened  if (mysqli\_connect\_errno()) {  echo "1: Connection Failed"; // error code #1 = connection failed  exit();  }  $username = mysqli\_real\_escape\_string($con, $\_POST["name"]);  $password = mysqli\_real\_escape\_string($con, $\_POST["password"]);  // Check for user's existence  $namecheckquery = "SELECT id, salt, hash FROM UserID WHERE username = '$username';";  $namecheck = mysqli\_query($con, $namecheckquery) or die("2: Name check query failed"); // error code #2 = name check query failed  if (mysqli\_num\_rows($namecheck) != 1) {  echo "5: Either no user with name, or more than one"; // error code #5 = number of name matching != 1  exit();  }  // Get login info from query  $existing\_info = mysqli\_fetch\_assoc($namecheck);  $salt = $existing\_info["salt"];  $hash = $existing\_info["hash"];  $loginhash = crypt($password, $salt);  if ($hash != $loginhash) {  echo "6: Incorrect Password"; // error code #6 = incorrect Password  exit();  }  // User is authenticated at this point, so let's get their score  $userId = $existing\_info["id"];  // Calculate the user's score based on transaction history  $scorequery = "SELECT SUM(PointsChange) AS points\_sum FROM Transaction WHERE UserID = '$userId';";  $scoreresult = mysqli\_query($con, $scorequery) or die("7: Score query failed"); // error code #7 = score query failed  $row = mysqli\_fetch\_assoc($scoreresult);  $pointsSum = $row['points\_sum'] ?? 0; // Use null coalescing operator to handle NULL result  $baseScore = 1000;  $totalScore = $baseScore + $pointsSum;  echo "0". "\t". $totalScore."\t". $userId;  ?> | |

#### Save Game API

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | <?php  $con = mysqli\_connect('localhost', 'root', 'root', '3D\_Connect\_4');  if (mysqli\_connect\_errno()) {  echo "1: Connection failed"; // error code #1 = connection failed  exit();  }  // Assuming IDs are passed directly through POST  $p1\_id = mysqli\_real\_escape\_string($con, $\_POST["p1\_id"]);  $p2\_id = mysqli\_real\_escape\_string($con, $\_POST["p2\_id"]);  $winner\_id = mysqli\_real\_escape\_string($con, $\_POST["winner\_id"]);  $moves = mysqli\_real\_escape\_string($con, $\_POST["moves"]);  // Begin transaction  mysqli\_begin\_transaction($con);  try {    // Insert into Session table  $insertSessionQuery = $con->prepare("INSERT INTO Session (WinnerID, Moves) VALUES (?, ?)");  $insertSessionQuery->bind\_param("is", $winner\_id, $moves);  $insertSessionQuery->execute();  $session\_id = $con->insert\_id;  // Insert into Game\_UserID for P1  $insertJunction1Query = $con->prepare("INSERT INTO Game\_UserID (UserID, SessionID) VALUES (?, ?)");  $insertJunction1Query->bind\_param("ii", $p1\_id, $session\_id);  $insertJunction1Query->execute();  // Insert into Game\_UserID for P2  $insertJunction2Query = $con->prepare("INSERT INTO Game\_UserID (UserID, SessionID) VALUES (?, ?)");  $insertJunction2Query->bind\_param("ii", $p2\_id, $session\_id);  $insertJunction2Query->execute();  if ($winner\_id === 0:  $reason = 'D';  $inertTransactionDrawQuery = $con->prepare("INSERT INTO Transaction (SessionID, UserID, PointsChange, Reason) VALUES (?, ?, ?, ?)");  $insertTransactionWinnerQuery->bind\_param("iiis", $session\_id, $ p1\_id, 0, $reason);  $insertTransactionWinnerQuery->bind\_param("iiis", $session\_id, $ p2\_id, 0, $reason);  else:  // Determine the loser ID  $loser\_id = ($winner\_id == $p1\_id) ? $p2\_id : $p1\_id;  // Insert into Transaction table for Winner  $pointsChange = -10;  $reason = 'W'; // Assuming 'W' for Win  $insertTransactionWinnerQuery = $con->prepare("INSERT INTO Transaction (SessionID, UserID, PointsChange, Reason) VALUES (?, ?, ?, ?)");  $insertTransactionWinnerQuery->bind\_param("iiis", $session\_id, $winner\_id, $pointsChange, $reason);  $insertTransactionWinnerQuery->execute();  // Insert into Transaction table for Loser  $pointsChange = 10;  $reason = 'L'; // Assuming 'L' for Loss  $insertTransactionLoserQuery = $con->prepare("INSERT INTO Transaction (SessionID, UserID, PointsChange, Reason) VALUES (?, ?, ?, ?)");  $insertTransactionLoserQuery->bind\_param("iiis", $session\_id, $loser\_id, $pointsChange, $reason);  $insertTransactionLoserQuery->execute();  // Commit transaction  mysqli\_commit($con);  echo "0"; // Success  } catch (mysqli\_sql\_exception $exception) {  mysqli\_rollback($con);  echo "3: Save game info failed - " . $exception->getMessage();  }  // Close all statement and connection  $insertSessionQuery->close();  $insertJunction1Query->close();  $insertJunction2Query->close();  $insertTransactionWinnerQuery->close();  $insertTransactionLoserQuery->close();  $con->close();  ?> | |

#### Retrieve Games Data API

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | <?php  $con = mysqli\_connect('localhost', 'root', 'root', '3D\_Connect\_4');  if (mysqli\_connect\_errno()) {  echo "1: Connection Failed"; // error code #1 = connection failed  exit();  }  $userId = mysqli\_real\_escape\_string($con, $\_POST["userid"]);  $queryGetGames = "  SELECT  gi.SessionID,  opp.username AS OpponentUsername,  (1000 + IFNULL((SELECT SUM(t.PointsChange) FROM Transaction t WHERE t.UserID = opp.id), 0)) AS OpponentScore,  (SELECT username FROM UserID WHERE id = gi.WinnerID) AS WinnerUsername,  CASE  WHEN gu.Colour = 1 THEN 'Black'  WHEN gu.Colour = 2 THEN 'White'  END AS UserRole,  JSON\_LENGTH(gi.Moves) AS TotalMoves  FROM  Game\_UserID gu  INNER JOIN Session gi ON gu.SessionID = gi.SessionID  INNER JOIN UserID usr ON gu.UserID = usr.id  INNER JOIN UserID opp ON opp.id <> usr.id AND opp.id IN  (SELECT UserID FROM Game\_UserID WHERE SessionID = gi.SessionID AND UserID <> gu.UserID)  WHERE  gu.UserID = $userId  ORDER BY  gi.SessionID DESC  LIMIT 10;  ";  $resultGetGames = mysqli\_query($con, $queryGetGames);  if (!$resultGetGames) {  echo "2: Game info retrieval query failed - " . mysqli\_error($con);  exit();  }  if (mysqli\_num\_rows($resultGetGames) > 0) {  $games = mysqli\_fetch\_all($resultGetGames, MYSQLI\_ASSOC);  echo json\_encode($games);  } else {  echo "3: No games found for the specified user.";  }    mysqli\_close($con);  ?> | |

#### Retrieve Move API

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <?php  $con = mysqli\_connect('localhost', 'root', 'root', '3D\_Connect\_4');  if (mysqli\_connect\_errno()) {  echo "1: Connection failed"; // error code for connection failure  exit();  }  if (!isset($\_POST["SessionID"])) {  echo "3: SessionID not provided"; // error code for missing SessionID  exit();  }  $SessionID = $\_POST["SessionID"]; // Missing semicolon added here  // Prepare the statement to select the Moves from GameInfo table for the given SessionID  $stmt = $con->prepare("SELECT Moves FROM Session WHERE SessionID = ?");  if ($stmt === false) {  echo "4: Query preparation failed"; // error code for preparation failure  exit();  }  $stmt->bind\_param("i", $SessionID); // Corrected typo here  $stmt->execute();  $result = $stmt->get\_result();  if ($row = $result->fetch\_assoc()) {  // Directly output the moves as the response  echo "0"."\t" . $row['Moves']; // 0 indicates success followed by the moves data  } else {  echo "2: No moves are found for the sessionID"; // error code for no moves found  }  $stmt->close();  $con->close();  ?> | |

# Function Table

|  |  |  |  |
| --- | --- | --- | --- |
| Function Name | Purpose | Return Type | Parameters |
| LoggedIn | Check if a user is logged in by determining if ‘username’ is not null | bool | None |
| LogOut | Logs out the current user by setting the ‘username’ to null | void | None |
| CallRegister | Initiates the registration process by starting the ‘Register’ coroutine | void | None |
| Register | A coroutine that send a registration requrest to a server. It posts the user’s name and password to a specificed URL and handles the response to determine success or failure of the registration | IEnumerator | None |
| VerifyInputs | Checks if the inputs in the ‘nameField’ and ‘passwordField’ meet certain condition (e.g., minimum length) and enables or disables the **submitButton** accordingly. | void | None |
| Start [Game UI] | Initialises the UI elements based on the user's login status. It updates the player display with the username if logged in, enables or disables the register and login buttons based on the login status, and sets the interactivity of the Online and Local buttons. | void | None |
| CallLogin | Initiates the login process by starting the **LoginPlayer** coroutine. | void | None |
| LoginPlayer | A coroutine that sends a login request to a server. It posts the user's name and password to a specified URL and processes the response. If successful, it updates the **DBManager** with the user's username and score and changes the scene to "GameUI". | IEnumerator | None |
| Update [Camera Rotation] | Called once per frame. It handles mouse input to rotate the camera around a target object. If the left mouse button is pressed, it starts tracking the mouse's position. If the button is held down, it calculates the direction of movement and rotates the camera around the target based on this movement. | void | None |
| OnMouseOver | Listens for mouse input when the cursor is over the GameObject this script is attached to. If the left mouse button is clicked, it logs the row and column of the column pressed, calls the **ColumnPressed** method on the **GameManager**, and adjusts the size and center of the BoxCollider component to visually represent the interaction. | void | None |
| Awake [GameManager] | Initialises the singleton instance of **GameManager**. | void | None |
| Start [GameManager] | Sets initial states for some UI elements, specifically deactivating the game over window. | void | None |
| SetUserDisplay | Updates UI elements with player names and scores. | void | string : usernameBlack,  string : usernameWhite  int : scoreBlack  int : scoreWhite |
| ColumnPressed | Handles input on a column press, sends the move to the server. | void | int : row  int : column |
| PlayBlock | Initiates the coroutine to animate and place a block in the playfield. | void | int : row  int : column  int : layer  int : currentPlayer |
| PlayBlockCourtine | Coroutine to animate block movement and update the playfield. | IEnumerator | int : row  int : column,  int : layer  int : currentPlayer |
| MoveToGoal | Moves a block towards its goal position during placement animation. | bool | Vector3 : goalpost  GameObject : block |
| WinCondition | Activates the game over UI elements based on the win condition. | void | bool : winner |
| OnRematchButton | Sends a rematch request to the server. | void | None |
| OnExitButton | Sends a game exit signal to the server and returns to the game UI. | void | None |
| Reset\_Physical\_Board | Cleans up the playfield, removing all block game objects. | void | None |
| Update [OutlineSelection] | Handles the highlighting and selection of objects under the mouse cursor. It enables or disables the **Outline** component on GameObjects to visually indicate highlight and selection. This method checks for mouse position and click to update the highlight and selection. | void | None |
| Awake [Playfield] | Initializes the singleton instance of **Playfield** and ensures it persists across scene loads. | void | None |
| ValidMove | Checks if a move is valid by finding an empty spot in a given column and row, starting from the top layer down. | int | int : row  int : column |
| DropBlock | Checks if a move is valid by finding an empty spot in a given column and row, starting from the top layer down. | void | int : layer  int : row  int : column  int : player |
| DebugBoard  [Playfield]  [GameRoom] | Creates a string representation of the current board state for debugging purposes. | string | None |
| Reset\_Board  [Playfield]  [GameRoom] | Resets the board to its initial state | void | None |
| OnClickOnHost | Initiates the connection setup and event registration for the client. | void | None |
| RegisterEvents  [Client]  [ClientNetworkHandler] | Registers the client-side event handlers for various network messages. | void | None |
| UnRegisterEvents  [Client]  [ClientNetworkHandler] | Unregister the client-side event handlers to prevent memory leaks or unwanted behaviour when the object is destroyed. | void | None |
| GetCurrentTeam | Returns the team ID assigned to the current client. | int | None |
| OnWelcomeClient | Handles the welcome message from the server, assigning the client's team. | void | NetMessage : msg |
| OnStartGameClient | Initiates the transition to the online game scene and sets up the game state based on the start game message. | void | NetMessage : msg |
| SetUpUI | Waits for the scene to finish loading and then updates the game UI with player information. | IEnumerator | NetStartGame : msg |
| OnMakeMoveClient | Updates the game state based on a move made by a player as broadcast by the server. | void | NetMessage : msg |
| OnRematchClient | Handles the rematch logic based on the rematch messages from players, updating the game state for a new match if both agree. | void | NetMessage : msg |
| OnEndGameClient | Processes the end game message, indicating the game's conclusion and the winning team. | void | NetMessage : msg |
| Start [NetworkManager] | Initialises the address input field by finding it in the scene and setting it up for use. | void | None |
| Awake [NetworkManager] | Ensures that there is only one instance of the **NetworkManager** throughout the scenes, making it a Singleton pattern. | void | None |
| OnOnlineHostButton | Initialises the server with a specific port and loads the "HostMenu" scene. | void | None |
| OnOnlineConnectButton | Initialises the client with the IP address provided in the address input field and a specific port, then loads the "HostMenu" scene. | void | None |
| OnHostBackButton | Shuts down the client connection, destroys the **NetworkManager** game object, and nullifies the instance to clean up before going back to a previous scene or state. | void | None |
| BackButtonClicked | A property to get the **backButtonClicked** state, indicating whether the back button was clicked. | bool | None |
| GoToRegister | Loads the scene for user registration. | void | None |
| GoToLogin | Loads the scene for user login | void | None |
| GoToMenu | Loads the scene for the main page. | void | None |
| GoToOnlineMenu | Loads the online menu scene, where users join an online game. | void | None |
| GoToOnlineFromHost | Handles navigation from the hosting setup back to the online menu, ensuring any network connections are properly shutdown. | Void | None |
| GoToReplayMenu | Loads the replay menu scene. | void | None |
| GoToReplay | Loads the scene where users can watch game replays. | void | None |
| GoToOnlineGame | Load the scene with the board where the game is played on. | void | None |
| OnClickOnHost | Prepares the server for hosting by registering network events and making the object persistent across scenes. | void | None |
| RegisterEvents | Subscribes to server-related network events to handle incoming messages appropriately. | void | None |
| UnregisterEvents | Unsubscribes from network events to prevent event handler leaks when the server is not active or the object is destroyed. | void | None |
| OnWelcomeServer | Handles the initial connection of a client, assigning them a team and notifying all clients in the room. It also sets up initial game conditions. | void | NetMessage : msg  NetworkConnection : cnn |
| OnMakeMoveServer | Processes move messages from clients, updates game state accordingly, and broadcasts the move to all clients in the same game room. | void | NetMessage : msg  NetworkConnection : cnn |
| OnRematchServer | Handles rematch requests from clients, resets the game state if both players agree to a rematch and broadcasts the rematch start. | void | NetMessage : msg  NetworkConnection : cnn |
| OnEndGameServer | Intended to handle game end scenarios, such as determining the winner and cleaning up game state. Currently, the implementation details are not provided. | void | NetMessage : msg  NetworkConnection : cnn |
| Start [GameEntry] | Begins the process of fetching the game history for the current user. | void | None |
| Awake [GameEntry] | Initialises the singleton instance of ‘GameEntry’ and sets it to persist across scene loads. | void | None |
| FetchGameHistory | Sends a request to a specified URL to retrieve the game history for a given username and processes the response to populate the UI. | IEnumerator | string : username |
| PopulateGameHistory | Instantiates UI elements for each entry in the game history and populates them with data. | void | List<GameData> gameHistory |
| OnGameEntryClick | Handles the event when a game entry is clicked. | void | GameData : data |
| CallReplay | Initiates the process for replaying a game based on the data from the clicked game entry. | IEnumerator | GameData : data |
| Awake [Replay] | Initialises the singleton instance of Replay | void | None |
| CallReplay | Starts fetching and processing move data for a given game session to replay it. | void | GameData : data |
| GetMove | Fetches move data from the server for he specified game session. | IEnumerator | GameData : data |
| DeserialiseJson | Converts a JSON string into a list of moves that can be replayed. | void | String : json |
| Update [Replay] | Listens for keyboard input (right and left arrows) to step through the replay forwards or backwards. | void | None |
| OnFowardArrow | Advances the replay to the next move. | void | None |
| OnBackArrow | Reverts the replay to the previous move. | void | None |
| PlayBlock | Initiates the placement of a block in the replay at the specified position. | void | int : row  int : column  int : layer |
| PlayBlockCoroutine | Coroutine to animate the block's placement during the replay. | IEnumerator | int : row  int : column  int : layer |
| RemoveBlock | Initiates the removal of a block in the replay, effectively undoing a move. | void | int : row  int : column  int : layer |
| RemoveBlockCoroutine | Coroutine to animate the block's removal during the replay. | IEnumerator | int : row  int : column  int : layer |
| MoveToGoal | Moves a block towards a target position, used in both placing and removing blocks. | bool | Vector3 : goalpost  GameObject : block |
| SwitchPlayer | Switches the current player indicator between Player 1 and Player 2. | void | None |
| Awake [Client] | Initialises the singleton instance of ‘Client’ | void | None |
| Init [Client] | Sets up the client's network driver and attempts to connect to a server at a given IP and port. | void | string : ip  ushort : port |
| Shutdown [Client] | Safely shuts down the network driver and marks the client as inactive. | void | None |
| OnDestroy [Client] | Invoked when the GameObject is being destroyed to ensure the network driver is properly disposed of. | void | None |
| Update [Client] | Updates the network driver, checks for connection status, and processes incoming network events. | void | None |
| CheckAlive [Client] | Checks if the client has lost connection to the server and invokes ‘connectionDropped’ action if so | void | None |
| UpdateMessagePump | Processes all incoming network events, handling connection, data reception, and disconnection | void | None |
| SendToServer | Serialises and sends a NetMessage to the server | void | NetMessage : msg |
| OnKeepAlive | Handles keep-alive messages by echoing them back to the server to maintain the connection | void | NetMessage : msg |
| Awake [Server] | Initialises the singleton instance of **Server**. | void | None |
| Init [Server] | Initializes the server, setting up the network driver and endpoint for listening to incoming connections. | void | ushort : port |
| Shutdown [Server] | Cleans up network resources when the server is no longer needed or is shutting down. | void | None |
| OnDestroy [Server] | Ensures network resources are cleaned up when the **Server** GameObject is destroyed. | void | None |
| Update [Server] | Regularly checks for new connections, processes incoming messages, and handles connection maintenance. | void | None |
| KeepAlive | Sends keep-alive messages to all clients to maintain active connections. | void | None |
| CleanupConnections | Removes any connections that are no longer active. | void | None |
| AcceptNewConnections | Accepts new incoming network connections. | void | None |
| AssignPlayerToRoom | Assigns a newly connected player to a game room based on matching criteria. | void | NetworkConnection : connection |
| UpdateUserInfo | Updates or adds the user information for a connected player/ | void | int : connectionid  int : userid  string : username  int score |
| UpdateMessagePump | Processes all incoming network events for each connection. | void | None |
| SendToClient | Sends a network message to a specific client. | void | NetworkConnection : connection  NetMessage : msg |
| Broadcast | Sends a message to multiple clients identified by their internal IDs. | void | NetMessage : msg  List <int> : playerIds |
| GetGameRoom | Retrieves the ‘GameRoom’ instance associated with a given room ID. | GameRoom | int : roomId |
| StartStoringMovesCoroutine | Starts a coroutine to store the moves of a completed game session. | void | List<int> : connections  string : jsonData  int : winner |
| StoreMove | Coroutine to send game session data to the backend | IEnumerator | List<int> : connections  string : jsonData  int : winner |
| GameRoom | Constructor that initialises the game room with a list of player IDs. | Constructor | List<int> : playerIDs |
| DropBlock | Places a block in the game board at the specified coordinates and checks for a win condition. | void | int : row  int : column  int : layer  int : player |
| WinCondition | Checks if the current move resulted in a win and handles the end of the game if so. | void | bool : winner |
| OnWinnerDetermined | Serialises and stores the game moves when a winner is determined. | void | None |
| WinCheck | Checks all possible win conditions to determine if the current player has won | bool | None |
| x\_axis\_Check | Checks for a win condition along the x-axis (horizontal). | bool | None |
| z\_axis\_Check | Checks for a win condition along the z-axis (vertical). | bool | None |
| z\_x\_axis\_Check | Checks for a win condition along diagonal lines in the x-z plane. | bool | None |
| y\_axis\_Check | Checks for a win condition along the y-axis (layer-wise vertical). | bool | None |
| x\_y\_axis\_Check | Checks for a win condition along diagonal lines in the x-y plane. | bool | None |
| z\_y\_axis\_Check | Checks for a win condition along diagonal lines in the z-y plane. | bool | None |
| x\_y\_z\_axis\_Check | Checks for a win condition along diagonal lines that span all three dimensions. | bool | None |
| SerialiseJson | Converts a list of moves into a JSON string for storage. | string | List<Vector3Int> : vectorList |
| SwitchPlayer | Alternates the turn between the two players. After a player makes a move, this function is called. | void | None |
| $insertuserquery | SQL query to insert user data into the database | None | string : $username  string : $hash  string : $salt |
| $namecheckquery | SQL query to check for duplication of username | string | string : $username |
| $queryGetGame | SQL query to get 10 most recent games | Int, string | int : $userId |
| $insertSessionQuery | SQL query to insert Session table | None | int : $winner\_id  string : $Moves |
| $insertJunctionQuery | SQL query to insert into the junction table for user table and session table. | None | int : player id  int : $session\_id |
| $insertTransactionWinnerQuery | SQL query to insert into Transaction for the winner of the session. | None | int : $session\_id  int : $winner\_id  int : $pointChange  string : $reason |
| $insertTransactionLoserQuery | SQL query to insert into Transaction for the loser of the session. | None | int : $session\_id  int : $loser\_id  int : $pointChange  string : $reason |

# Data Table

|  |  |  |
| --- | --- | --- |
| Field Name | Purpose | Data Type |
| username | Stores the username of the currently logged-in user | string |
| Score | Stores the score of the currently logged-in user | int |
| nameField | Input field for the user’s name | string |
| passwordField | Input field for the user’s password | string |
| submitButton | A button that ttriggers the registration process | Button |
| registerButton | A button in the UI for the user to navigate to the registration screen. | Button |
| loginButton | A button in the UI for the user to navigate to the login screen. | button |
| OnlineButton | A button that allows the user to start or join an online game. Its interactivity is dependent on the user's login status. | Button |
| LocalButton | A button that allows the user to start a local game. It is always interactive regardless of the user's login status. | Button |
| playerDisplay | A text field displaying the player's username if logged in. | string |
| cam | The camera that will be rotated. | Camera |
| target | The target object around which the camera will rotate. | Transform |
| previousPosition | Stores the previous position of the mouse. This is used to calculate the direction of rotation. | Vector3 |
| Row [ColumnInput] | Identifies the row of the game element this script is attached to. (Preassigned) | int |
| column [ColumnInput] | Identifies the column of the game element this script is attached to. (Preassigned) | int |
| Instance  [GameManager]  [Playfield]  [NetworkManager]  [GameEntry]  [Replay]  [Client]  [Server]  [GameEntry] | Singleton instance of the class. | GameManager  Playfield  NetworkManager  GameEntry  Replay  Client  Server  GameEntry |
| sceneManagerNavigation | Utility for scene navigation, not explicitly used in the provided code. | SceneManagerNavigation script |
| instantiatedBlocks | Tracks all block game objects instantiated during gameplay. | List<GameObject> |
| b\_block | Prefab for a black block. | GameObject |
| w\_block | Prefab for a white block | GameObject |
| startPoint | The starting point for block placement. | Transform |
| gameOverWindow | UI element displayed when the game is over. | GameObject |
| userNameDisplay | UI element for the child within it. (i.e. blackPlayerName and whitePlayerName) | GameObject |
| blackPlayerName | UI text for displaying the black player's name. | string |
| blackPlayerScore | UI text for displaying the black player's score. | string |
| whitePlayerName | UI text for displaying the white player's name. | string |
| whitePlayerScore | UI text for displaying the white player's score. | string |
| rematchIndicator | UI element to indicate for a rematch option for client. | Transform |
| highlight | Tracks the currently highlighted object. | Transform |
| selection | Tracks the currently selected object. | Transform |
| raycastHit | Stores Information about the Raycast hit, used to determine the object under the mouse cursor. | RaycastHit |
| rows | Number of rows in the game board. | int |
| columns | Number of columns in the game board. | int |
| layers | Number of layers in the game board. | int |
| winlength | Number of consecutive blocks needed to win. | int |
| board | Stores the current state of the game baord | int[,,] |
| currentTeam | Stores the teamID assigned to this client, indicating whether they are player 1 or 2. | int |
| playerRematch [ClientNetworkHandler]  [ServerNetworkHanlder] | Tracks the rematch status for each player to determine if both players have agreed to rematch. | bool[] |
| server | Reference to the server component attached to the same GameObject. | Server script |
| client | Reference to the client component attached to the same GameObject. | Client script |
| addressInput | The input field where the user enters the IP address for connecting to a server | TMP\_InputField |
| backButtonClicked | Tracks whether the back button has been clicked to navigate away from the current scene. | bool |
| GameDataList | Holds a list of ‘GameData’ instances. | Class |
| GameData | Represents the data for a single game session. | Struct |
| GameEntryPrefab | Prefab used for each game entry in the UI. | GameObject |
| contentPanel | The parent transform for game entry UI elements. | Transform |
| b\_block | Prefab for black block. | GameObject |
| w\_block | Prefab for white block. | GameObject |
| startPoint | The starting point for block placement in the replay. | Transform |
| P1\_User | Text component for displaying Player 1’s username and score. | TMP\_Text |
| P2\_User | Text component for displaying Player 2's username and score. | TMP\_Text |
| Winner | Text component for displaying the winner of the game. | TMP\_Text |
| instantiatedBlocks [Replay] | A list of instantiated block GameObjects for the current replay. | List<GameObject> |
| moves | A list of moves made during the game, used for replaying the game. | List<Vector3Int> |
| currentPlayer | Indicates which player's move is currently being replayed. | int |
| isKeyPressed | Flag to prevent processing the same key press multiple times. | bool |
| pointer | Index to keep track of the current move in the replay | int |
| driver [Client] | The network driver used for managing network connections and events. | NetworkDriver |
| Connection | Represents the network connection to the server. | NetworkConnection |
| isActive [Client] | Indicates whether the client is currently active and connected. | bool |
| connectionDropped | Action to be invoked when the connection to the server is lost. | Action |
| rooms | Tracks which players by their internalIds are in which game rooms. | Dictionary<int, List<int>> |
| gameRooms | Stores ‘GameRoom’ objects for managing the state of each game session. | Dictionary<int, GameRoom> |
| userInfo | Stores user information (userid, username, score) for each connected player. | Dictionary<int, Userinfo> (int : Internalid) |
| roomCount | Tracks the number of active game rooms. | int |
| driver [Server] | The network driver used for low-level network operations. | NetworkDriver |
| connections | A list of active network connections. | NativeList<NetworkConnection> |
| isActive [Server] | Indicated whether the server is actively listening for connections. | bool |
| keepAliveTickRate | The interval at which KeepAliveMessages are sent. | const float |
| lastKeepAlive | Tracks the last time a keep-alive message was sent. | float |
| connectionDropped | Action Invoked when a connection is lost. | Action |
| coordinateList | Stores the coordinates of each move made during the game, so that it can be stored in the database. | List<Vector3Int> |
| currentPlayer | Indicates which player’s turn it is. | int |
| connectionID | List of connection IDs of the players in the game room. | List<int> |
| $con | Represents the database connection Object | Object |
| $username | Stores the username obtained from the client’s request | string |
| $password | Stores the password obtained from the client’s request | string |
| $salt | Generates a salt used n password hashing | string |
| $hash | Stores the hashed password | string |
| $namecheck | Results of the username check query | string |
| $existing\_info | Array containing user information fetched from the database | List |
| $loginhash | Hashed password for comparison during login | string |
| $userId | Stores the user ID after successful authentication | int |
| $scoreresult | Result of the score calculation query. | int |
| $row | Array containing the result of the score query. | List |
| $pointSum | Total points earned by the user. | int |
| $baseScore | Base score value. | int |
| $totalScore | Total score including base score and points earned. | int |
| $SessionID | Stores the session ID obtained from the client's request. | int |
| $stmt | Prepared statement object for executing SQL queries. | Object |
| $result | Result set obtained from executing the SQL query. | Object |
| $p1\_id | Player 1's ID obtained from the client's request in the game session script. | int |
| $p2\_id | Player 2's ID obtained from the client's request in the game session script. | int |
| $winner\_id | Winner's ID obtained from the client's request in the game session script. | int |
| $moves | Game moves obtained from the client's request in the game session script. | int |

# References:

<https://kevurugames.com/blog/unity-vs-unreal-engine-pros-and-cons/>

<https://www.gamedesigning.org/career/video-game-engines/>

<https://cloudinfrastructureservices.co.uk/lamp-vs-xampp-vs-wamp-vs-mamp-whats-the-difference/>

<https://rocketbrush.com/blog/maya-vs-blender-for-3d-game-artist/>

<https://docs.unity.com/ugs/manual/game-server-hosting/manual/concepts/builds/>

<https://www.red-gate.com/simple-talk/databases/sql-server/performance-sql-server/designing-highly-scalable-database-architectures/>

<https://www.guru99.com/sql-vs-nosql.html>

<https://docs.unity3d.com/Packages/com.unity.transport@0.3/manual/index.html>

<https://www.youtube.com/watch?v=SLgZhpDsrfc>

<https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

<https://www.javatpoint.com/mini-max-algorithm-in-ai>

<https://www.geeksforgeeks.org/delegation-vs-inheritance-java/>