Multiplayer Connect 4 Game

Computer Science NEA

Name: *Alex Fairley*
Candidate Number: *5849*

Centre Name: Barton Peveril College
Centre Number: 58231

0 Contents

1 Analysis

1.1 Statement of Problem

Connect 4 is a classic board game, which takes a relatively simple concept and turns it into an enjoyable 2-player game for anyone and everyone. However, due to current *global* circumstances, specifically the Covid-19 virus and the ensuing lockdowns which have impacted *billions* of people around the world, many have been unable to enjoy the game with family and friends. There are currently few to no decent solutions available for allowing people to experience the game online, which would coincide with social distancing guidelines while enabling people to play with their friends/family, and enjoy themselves while doing it. The game is short, entertaining and makes for a great distraction from the miserable state of life at the moment, encouraging continued connection with loved ones. *Image below from Amazon:*

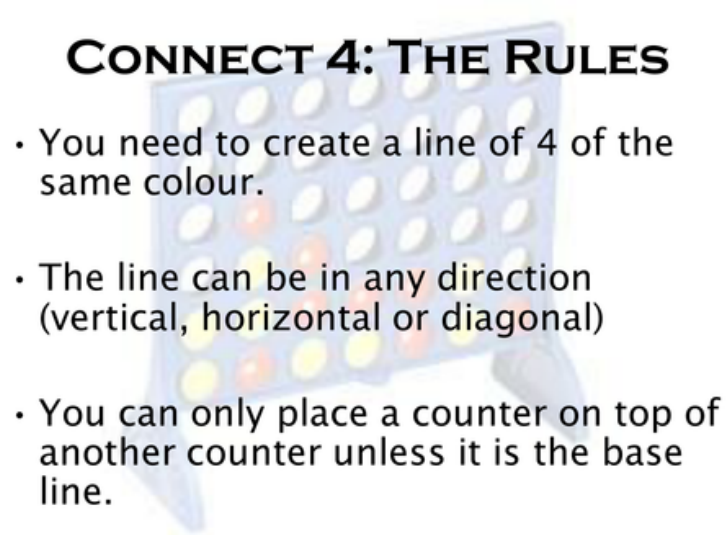
My aim for this project is to create an online version of Connect 4, which consists of local play and multiplayer capabilities, as well as an AI mode as well. This should provide an enriching experience for the user(s), bringing a beloved board game to their computers, while accounting for the distressing circumstances which currently affect them.

1.2 Background

Connect 4 is a 2-player board game that came out in 1974. It consists of a blue board, which had holes in it of dimensions (7x6, CxR) and also two sets of counters, coloured red and yellow exclusively. The objective of the game is to achieve a line of 4 counters, of your colour, which can be of any orientation

(horizontal, vertical or diagonal). This line cannot be broken by anything, be it an opposing player's counter or an empty space. Additionally, counters must be dropped from the top of each of the 7 rows, where they must fall until they reach either the bottom of the board or the token/stack of tokens that sits above the bottom. This adds a layer of complexity to the game, which can open up strategic play. The Player's alternate goes until either one achieves the goal of 4 in a row, or the board is full with no winner, which ultimately concludes as a draw.

Image below from tes.com



My online version will incorporate all of these features, albeit when appropriate, to bring the game seamlessly to the electronic world, where it can be enjoyed between 2 players, or even by one's self. Either way, it should be entertaining, the multiplayer modes should be engaging and the AI should be challenging. By combining all of these my project will successfully recreate Connect 4, alongside modes which will make the game as enjoyable as it is in-person.

1.3 End-User

My program's end-user is aimed at all/anyone who wants to be able to experience connect 4 as a virtual game. It will be played by a variety of interested users, whether they already understand and enjoy connect 4, and just want to experience it online, or if it is someone completely new to the game, who wants to try it out. This extends to any way they want to play it, be it single-player, local multiplayer, or online multiplayer.

As it is a relatively simple yet inviting game, it can be enjoyed by all age groups and individuals. The program should be easy to access, simple to navigate, and overall convenient for the user.

For my interviews, I'll be interviewing two current students, aged 17, who have an interest in the game. I'll be questioning them on potential features and also any recommendations/preferences they may have.

1.4 Initial Research

1.4.1 Existing, similar programs

Similar Program 1: https://www.mathsisfun.com/games/connect4.html

This first program is found on a free website called mathisfun, a relatively simple website that can be found as one of the top results when searching for "Connect 4 Online" through google. It consists of a local multiplayer mode, where two players can play against each other on the same computer, as well as an AI, which consists of 4 difficulties (Too Easy, Easy, Medium, and Hard).

The design of the game is very simple, which is to be expected from connect 4, and the layout is easy to navigate. It is simple to replay games and the AI acts instantaneously. A small message is displayed near the top of the program demonstrating who's turn it is and eventually, the winner of the match.

Pros:

- Local Multiplayer and also Vs AI mode.
- Simple and easy to navigate UI.
- Multiple difficulties of AI.
- Individual players can change their names as appropriate.
- Easy to play consecutive games, and a tally for respective wins is kept.

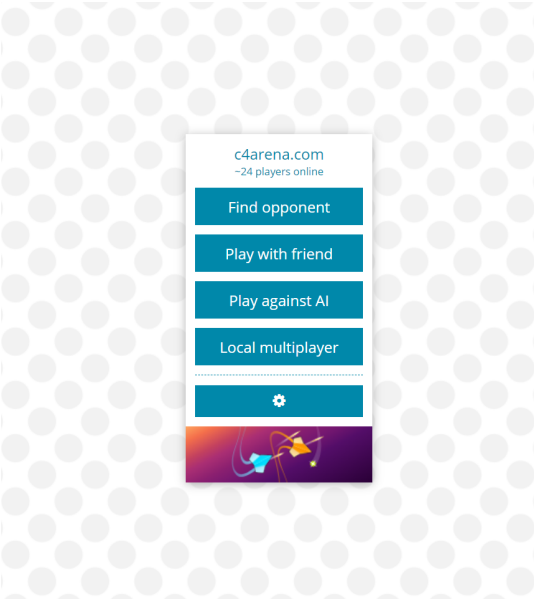
Cons:

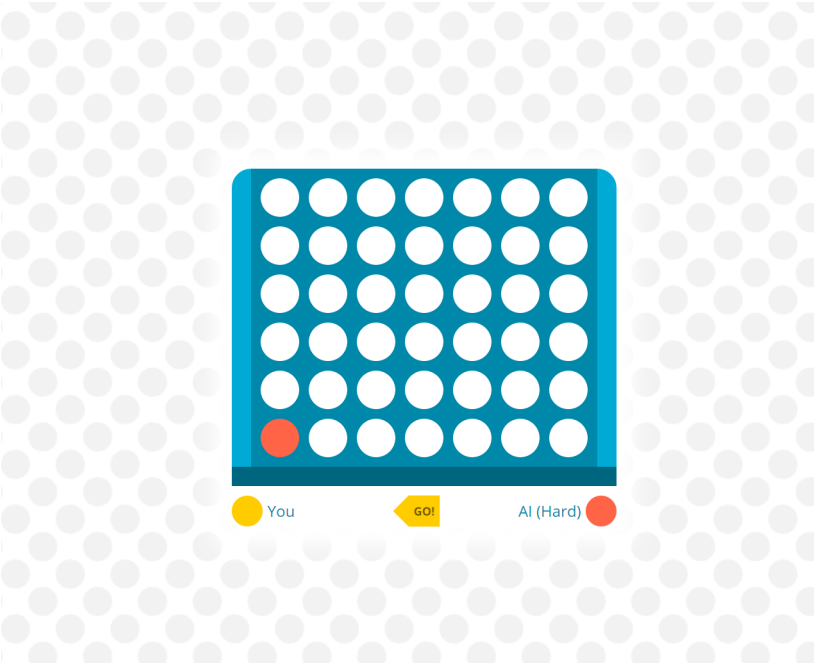
- The webpage runs advertisements.
- "*Too Easy*" difficulty is completely unnecessary, the AI feels suicidal and misses basic moves, making it completely obsolete when an "*Easy*" difficulty already exists.
- Lack of online multiplayer.
- Requires an internet connection to play, even though there are no online aspects (This is due to it being a web-based application).

In Conclusion, while I do enjoy the program as a simple variant of the game, in reference to the UI and playability, the drawbacks from the lack of an online multiplayer, as well as a WiFi connection being mandatory due to the nature of web-based applications make it a relatively lack-lustre option. For a user who just wants to play a quick game against a decent AI, or even against an associate on the same computer, then this application will serve its purpose. However, for those looking for more variety, and options extended towards online capability or the potential for offline play, this would be less than suitable.

Similar Program 2: https://c4arena.com/

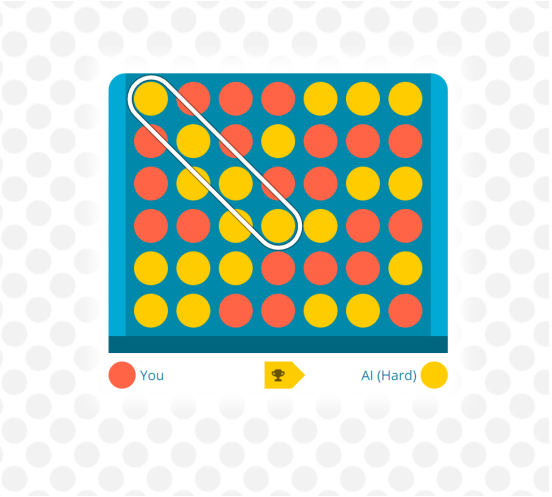
The second program is also a free, web-based application called c4arena, which has an appealing UI and consists of all the features which I am aiming to include within my program (Local play, Online play, and single-player vs AI).

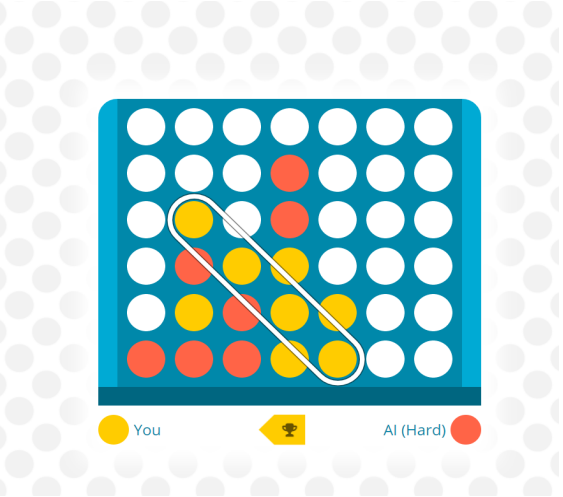




However, the AI is incredibly simple, with the hardest difficulty I was able to beat it on the first try, and it plays in a much more unorthodox way than other AI's which take advantage of algorithms such as MinMax (As an example, more on algorithms in the next section).

Intriguingly, after several playthroughs against the "*Hard*" difficulty of the AI, it would move very differently at the start of games, even when the player would make identical moves. I can only assume that the AI has been programmed with multiple different, randomly chosen, starting movesets, which, aside from preventing the user from achieving 4 in a row and winning, are prioritised over any other move. This is fascinating because, in some instances, the starting moves would be extremely weak, making it exceptionally easy to beat, and yet in others, could cause the AI to play almost flawlessly and be incredibly difficult to beat. Two screenshots provided below demonstrate two separate games I undertook vs the "*Hard*" AI.





The one on the right represents one where the AI played so poorly, it allowed me to win in only a few moves, and yet in the other, beat me through the last move in the game. Both games saw me make the starting move, where I placed a counter down the centre of the board.

This suggests to me that the AI utilises a learning algorithm with some predetermined moves hardcoded into the AI to try to identify the correct moves to make. However, this can lead to bad decision making by the AI, as it prioritises sometimes incorrect and even fatal moves.

Pros:

- Incorporates a variety of game modes for a variety of user needs.
- Well-designed user interface and game board.
- Multiple AI difficulties.
- Multiplayer for random opponents AND friends.

Cons:

- AI is incredibly inconsistent, which is extremely bad for an AI.
- Lack of online players makes the random opponents' mode semi-redundant.
- Also requires an internet connection (web-based)
- No scoring system for consecutive games-played

Overall, I think this program is excellent for those seeking only the multiplayer aspects to play with friends. The local and online multiplayer game modes are well made and fun to play, largely due to the graphic design. However, the AI is an extreme weakness as it is poorly designed, leading to it being unenjoyable due to the inconsistency. As a program intended for two friends to play against each other, be it online or local, it is superb. The random-opponent function, while a great concept, is largely wasted due to a lack of players. On the other side, though, when it comes to singleplayer it simply fails as a tangible application for user's to enjoy.

1.4.2 Potential abstract data types/algorithms

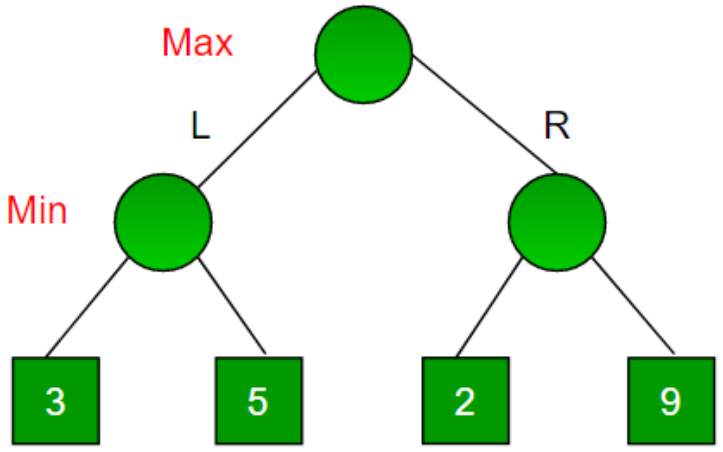
For the AI portion of my project, I will be incorporating a few solving algorithms to make the AI as capable as I am able. Below is a look at a few algorithms I

can potentially use to construct my AI, and I will likely utilise one of or a combination of these.

Algorithm 1 - MinMax:

The MinMax algorithm is an algorithm that aims to maximise the minimum possible gain from any given decisions between two parties. To expand on this, imagine there were two individuals, one whose aim is to maximise the total number (outcome) and the other aims to minimise the total number (outcome). If you presume that each individual always makes the best possible decision for themselves, it is possible to consider all possibilities and therefore arrive at the best number for yourself, i.e maximising the minimum possible gain. Here is a diagram to help better explain this: (Please bear in mind this is a heavily simplified representation of MinMax)

Image below from GeeksforGeeks.org



Looking at the bottom, we can see the leaf nodes as all the possible outcomes from any decisions made, represented by the numbers housed in squares. Assuming that the maximiser is going to make the final move, i.e picking the end leaf node, as a minimiser we want to minimise the maximum gain. So looking at the bottom we can see that the best number for the maximiser would be 9. Therefore, we avoid traversing right and instead traverse left, to where the highest number is 5, therefore minimising the maximum gain of the maximiser.

If the positions were reversed, and the minimiser was picking last, we can see the most ideal number for them is 2. Therefore, as the maximiser, we want to maximise the minimum gain, and so we traverse left, where the lowest

possible number is 3, therefore achieving a total 1 higher than if we traversed right.

So, incorporating this to connect 4 is easy to conceptualise, but quite hard to implement. Connect 4 is simple to understand in a min-max perspective, as each player plays the game turn by turn and each has an opposite objective, reaching 4 in a row with their respective tokens. It's about taking the best path possible to reach 4 in a row. Therefore on one particular turn, a player attempts to maximise their "score" subsequently minimising their opponent's. This can be conceptualised as a tree, such as the one above, except the depth of the tree would be incredibly more complicated, as technically every single possible move would be considered. As there are 7 moves available to each player at any given time, due to the nature of dropping a token into a column, of which there are 7. This means each move the AI considers further into the future, the number of potential outcomes grows by exponents of 7. For example, if we wanted our AI to consider 3 moves into the future, the total number of possible outcomes would be 7^3 = 343. Having a computer traverse this number of moves is relatively easy, but also only considering 3 moves into the future would make a relatively poor AI.

So how many moves ahead does an AI have to consider for it to be successful in its purpose? Well, although it is subjective, we can look at some other games to see how many moves ahead of their best players look. In chess, it is estimated that grandmasters look ahead by about 8 moves, which likely means that if our AI could look ahead 8 moves it would be successful. However, looking 8 moves ahead means our AI has to consider 7^8 = 5764801 possible outcomes. On each turn. This will be incredibly hard for a computer to process but also would make for a much more successful AI. Considering how many moves to look ahead is something I believe I will need to consider when it comes to the prototype, as it is very dependent on processing power and the speed at which the AI can search through each possible outcome. There are also a few different ways in which the algorithm can be applied, such as how board states are scored. Would having more tokens in a row be scored higher, or would certain board states be more valuable than others? These are just a few ways of implementing MinMax into Connect 4.

Algorithm 2 - Alpha-Beta Pruning

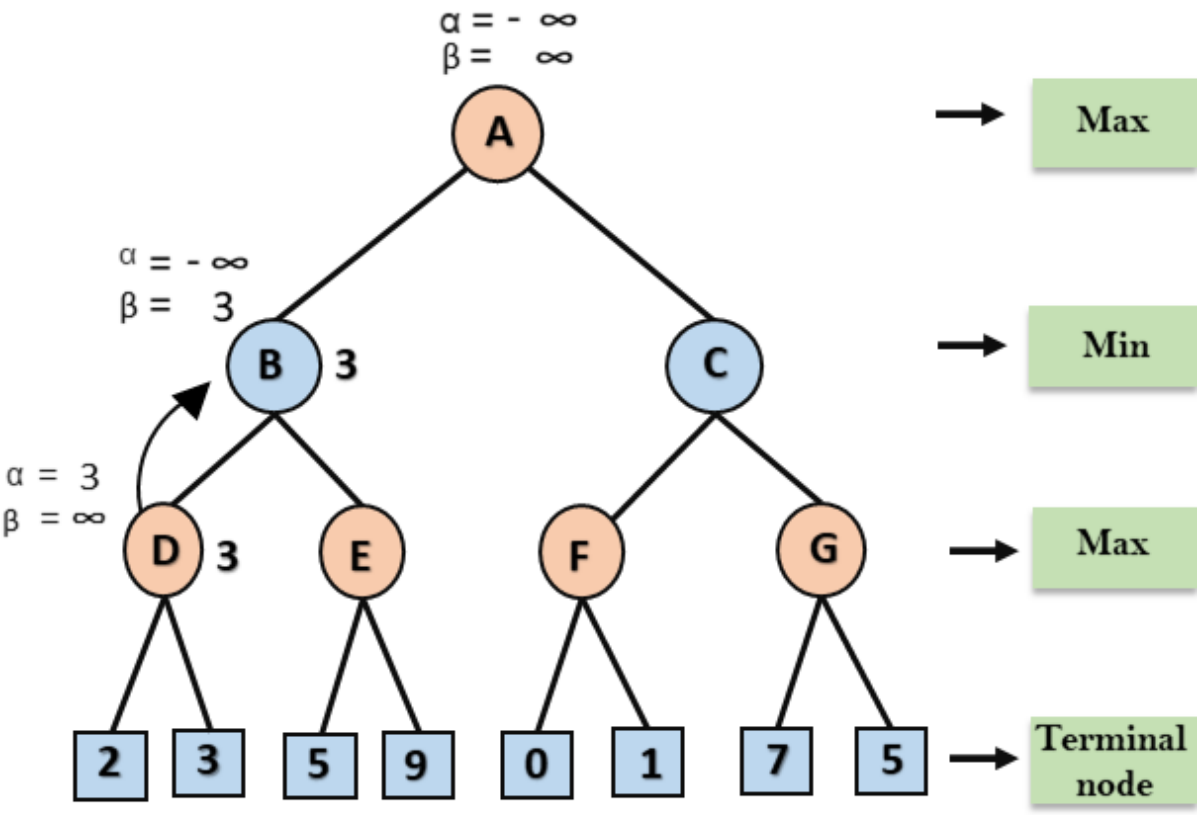


Image above from Javatpoint.com

Alpha-Beta pruning, in a generic sense, is effectively a way of increasing the efficiency of a searching algorithm by removing any options which are never going to be chosen by the algorithm. For example, in Connect 4 you are never going to intentionally make a move, which would cause you to lose on your opponents next turn. Alpha-Beta pruning ensures the searching algorithm ignores these options, improving the speed at which the algorithm can run. This algorithm is often used in conjunction with the MinMax algorithm, as they complement each other perfectly and this is why I have included it within my possible options for an algorithm.

An easier way of conceptualising this is imagining you are reviewing all your possible moves in a linear way. If each move had a value attributed to it,

evaluating how good of a move it is, why would you consider any moves with a lower value than the one you had found previously? You just wouldn't, and this is where alpha-beta pruning would immediately reject any worse moves than a move that has already been identified. This prevents them from being fully evaluated as an option, therefore saving time for the AI. Applying this to a Connect 4 AI, say a move could be made which would guarantee a loss in the next 2 moves, irrespective of what moves were made by the AI after. You would not consider this move, or any moves after this one, as they will never be played. So, instead of considering each one in-depth, they can all be ignored.

1.4.3 First interview

The interview was conducted over a live discord call. The responses below are the transcribed answers to my questions from two separate interviewees.

Q1. Do you think a leaderboard system is important? If so, would you prefer it to be global or local?

Interviewee 1: Yes, a leaderboard would be a great feature to implement, it would add a competitive nature to the program. Depending on the difficulty, a global leaderboard would be the better option as you can compare with a wider range of people, however, a local leaderboard is still a viable option as it still acts as a personal journal of your progress in the game.

Interviewee 2: Yes I believe that a leader board is important as it would give a stronger sense of goal and accomplishment, this is good as playing connect four multiple times can get repetitive and boring with no goal in mind when it comes to local or global I don't have a preference as both would give the ability to track progress and give a goal.

Q2. Would you prefer a single AI, which has a set difficulty, or would you like the ability to choose between varying difficulties? (Easy, Medium, Hard etc)

Interviewee 1: Being able to set the difficulty of an AI before the game would be a great feature, it would mean you can improve and develop yourself further instead of playing against a linear difficulty AI.

Interviewee 2: I think that having multiple difficulties is important as it gives the ability to set a goal and watch as you improve as well as it is more fun and accessible for all players no matter what the skill.

Q3. Do you think there's a particular mode that is more important for you or any other users? For example, multiplayer over AI etc.

Interviewee 1: Multiplayer would be the preferred mode out of the two, the ability to compete with friends and other people is a great feature of a game.

Interviewee 2: For me, the most important mode would be multiplayer as normally you cannot play connect 4 with your friends unless you are in the same room, however with online multiplayer we can play when we are not together.

Q4. Is a multi-game scoring system important? So the program would have a running score for each player so that multiple games can be played and the overall score would be displayed?

Interviewee 1: That is a very good feature to add, you can make game modes in the multiplayer section like "Best to 3", or "Best out of 5" to add some more competitive games to the multiplayer.

Interviewee 2: Yes I believe this is important as it adds a more competitive nature to the game which is a feature that is needed when playing with your friends.

Q5. Do you have any particular features from other online connect 4 games or similar which you think would be beneficial to incorporate?

Interviewee 1: As I mentioned, a leaderboard that keeps a track of scores against either friends or even the AI, many connect 4 games tend to not implement this and it could be a great feature to add.

Interviewee 2: A feature which would be nice to see added would be the overall ranking system from Chess.com, giving you the ability to gauge a player's skill from their ranking.

Evaluation after the interview:

So after conducting the interviews with two potential end-users, I have learnt a few things. Clearly, some form of a leaderboard would be greatly preferred by both of these end-users, and so is definitely a feature I should look to implement in my program. Additionally, when it comes to the AI having multiple difficulties would be useful as it makes it more useful for “all players” which is something I agree with, and so is something I will also look to implement. It looks like more of my time should be dedicated to ensuring the multiplayer mode is as good as possible, due to the interaction this allows between friends. It seems the scoring system is desirable in a similar fashion to the leaderboard, and so I may look to find a way of combining these features when I integrate them. This lines up with the answers to question 5, where it seems both individuals were really interested in a leaderboard system, meaning this should definitely be a key feature I want to include in my program.

1.4.4 Key requirements

AI Single Player Mode: This will be a mode where a user plays against an AI at Connect 4, the AI should have at least a couple of different difficulty levels to appeal to players from a variety of skill levels.

Local Multiplayer: This should be a game mode where two players, who are both using the same computer, should be able to play against each other. This means that both are using the same controls to play, and they will just alternate turns.

Online Multiplayer: Similar to the local multiplayer, except across the internet, allowing two players on separate computers and networks to play the game. Will require a server-client connection method such as port-forwarding.

Leaderboard: From my interview, it is clear that a leaderboard is a heavily desired feature for my game, and so therefore should definitely be a key feature I implement. Should likely be a global leaderboard for all users. Should also include a live scoring system for when two players are playing against each other.

1.5 Further Research

1.5.1 Prototype

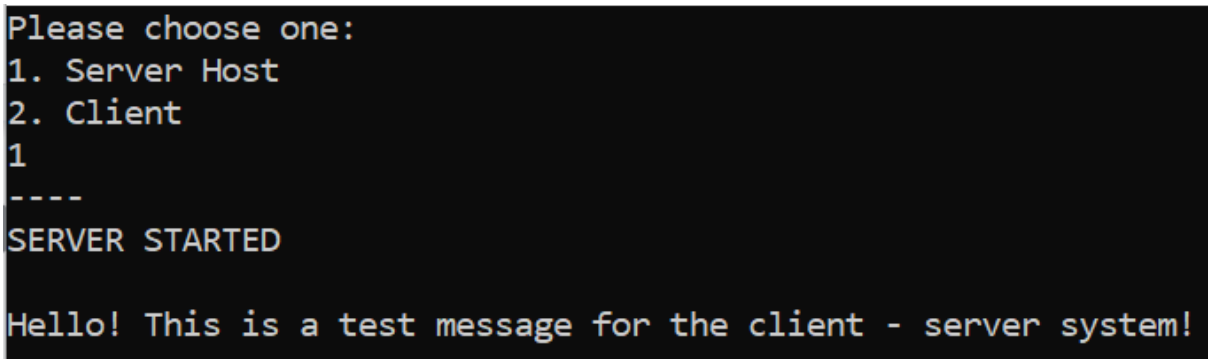
Prototype 1: Client - Server

Code:

```
Module Module1
    Dim Server As ServerTCP
    Dim Client As ClientTCP
    Sub Main()
        NetworkMenu()
        Console.ReadKey()
    End Sub
    Sub NetworkMenu()
        Dim choice As Integer
        Console.WriteLine("Please choose one:")
        Console.WriteLine("1. Server Host")
        Console.WriteLine("2. Client")
        choice = Console.ReadLine()
        Select Case choice
            Case 1
                Server = New ServerTCP
                Console.WriteLine("----")
                Console.WriteLine("SERVER STARTED" & vbCrLf)
                AddHandler Server.Received, AddressOf OnlineRecieved
            Case 2
                Client = New ClientTCP("86.3.155.208", 1234)
                If Client.Client.Connected Then
                    Console.WriteLine("Connected")
                End If
                Do
                    Console.WriteLine("Please enter a test message:")
                    Client.Send(Console.ReadLine)
                Loop
        End Select
    End Sub
    Sub OnlineRecieved(Sender As ServerTCP, Message As String)
        Console.WriteLine(Message)
    End Sub
End Module

Imports System.IO
```

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Threading
Public Class ServerTCP
    Public Event Received(sender As ServerTCP, Message As String)
    Public ServerIp As IPAddress = IPAddress.Parse("192.168.0.56")
    Public ServerPort As Integer = 1234
    Public Server As TcpListener
    Private CommThread As Thread
    Public IsListening As Boolean = True
    Private Client As TcpClient
    Private ClientMessage As StreamReader
    Public Sub New()
        Server = New TcpListener(ServerIp, ServerPort)
        Server.Start()

        CommThread = New Thread(New ThreadStart(AddressOf Listening))
        CommThread.Start()
    End Sub
    Private Sub Listening()
        Do Until IsListening = False
            If Server.Pending = True Then
                Client = Server.AcceptTcpClient
                ClientMessage = New StreamReader(Client.GetStream)
            End If
            Try
                RaiseEvent Received(Me, ClientMessage.ReadLine)
            Catch ex As Exception

            End Try
            Thread.Sleep(100)
        Loop
    End Sub
End Class

Public Class ClientTCP
    Public Client As TcpClient
    Public MessengerStream As StreamWriter
    Public Sub New(IpAddress As String, Port As Integer)
        Client = New TcpClient(IpAddress, Port)
        MessengerStream = New StreamWriter(Client.GetStream)
    End Sub
    Public Sub Send(Message As String)
        MessengerStream.Write(Message)
        MessengerStream.Flush()
    End Sub
End Class
```

Screenshots:



Prototype 2: Game Example

Code:

```
Module Module1
    Dim Board(2, 2) As String
    Sub Main()
        Do
            For i = 0 To 2
                For j = 0 To 2
                    Board(i, j) = " "
                Next
            Next
            Menu()
            Threading.Thread.Sleep(2000)
            Console.Clear()
        Loop
        Console.ReadKey()
    End Sub

    Sub Menu()
```

```
        Console.WriteLine("Please choose a Gamemode:")
        Console.WriteLine("1. Local Play")
        Select Case Console.ReadLine()
            Case 1
                Console.Clear()
                Dim token As String
                Console.WriteLine("Player 1: Please choose a token (O or
X)")
                token = UCase(Console.ReadLine())
                Console.WriteLine("Please enter a username:")
                Dim Player1 As New Player(token, Console.ReadLine())
                Console.WriteLine("Player 2: You'll be attributed the
other token")
                Select Case token
                    Case "O"
                        token = "X"
                    Case "X"
                        token = "O"
                End Select
                Console.WriteLine("Please enter a username:")
                Dim Player2 As New Player(token, Console.ReadLine())
                Console.Clear()
                LocalPlay(Player1, Player2)
        End Select
    End Sub

    Sub LocalPlay(Player1 As Player, ByVal Player2 As Player)
        Dim check As Boolean
        Do
            check = False
            Console.WriteLine(Player1.name & ", it is your turn, please
select a square: ")
            Console.WriteLine(" ")
            DisplayBoard()
            Console.WriteLine(" ")
            If Player1.token = "X" Then
                Console.ForegroundColor = ConsoleColor.Red
            ElseIf Player1.token = "O" Then
                Console.ForegroundColor = ConsoleColor.Yellow
            End If
            TraverseBoard(Player1)
            Console.ForegroundColor = ConsoleColor.White
            Console.Clear()
            If isWon() = True Then
                Player1.won = True
                Exit Do
            End If
```

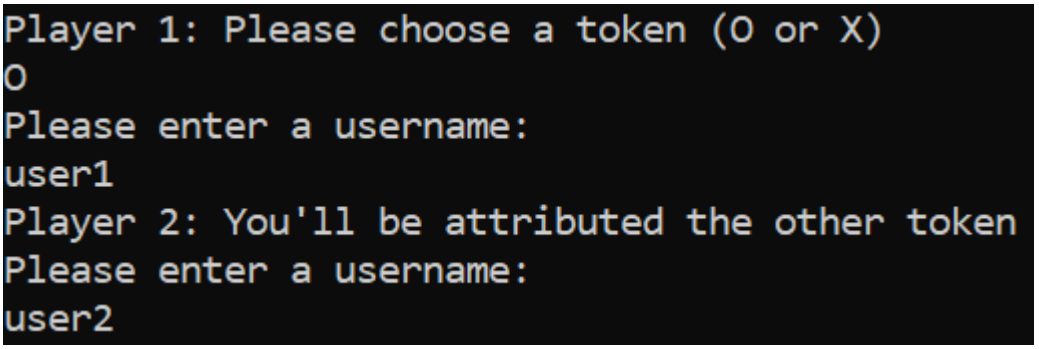
```
            For i = 0 To 2
                For j = 0 To 2
                    If Board(i, j) = " " Then
                        check = True
                    End If
                Next
            Next
            If check = False Then
                Exit Do
            End If
            Console.WriteLine(Player2.name & ", it is your turn, please
select a square: ")
            Console.WriteLine(" ")
            DisplayBoard()
            Console.WriteLine(" ")
            If Player2.token = "X" Then
                Console.ForegroundColor = ConsoleColor.Red
            ElseIf Player2.token = "O" Then
                Console.ForegroundColor = ConsoleColor.Yellow
            End If
            TraverseBoard(Player2)
            Console.ForegroundColor = ConsoleColor.White
            Console.Clear()
            If isWon() = True Then
                Player2.won = True
                Exit Do
            End If
            For i = 0 To 2
                For j = 0 To 2
                    If Board(i, j) = " " Then
                        check = True
                    End If
                Next
            Next
        Loop Until check = False
        If Player1.won = True Then
            WinScreen(Player1)
        ElseIf Player2.won = True Then
            WinScreen(Player2)
        Else
            Console.WriteLine("DRAW! NOBODY WINS!")
        End If
        Console.WriteLine(" ")
        Console.WriteLine("Returning to menu...")
    End Sub

    Sub WinScreen(winner As Player)
```

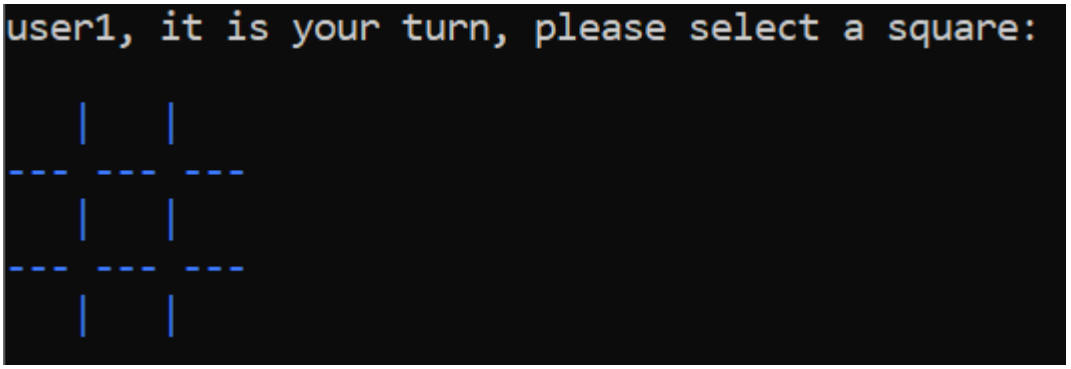
```
        Console.WriteLine(winner.name & " IS THE WINNER!")
    End Sub

    Sub TraverseBoard(Current As Player)
        Console.SetCursorPosition(1, 2)
        Dim press As ConsoleKey
        Dim row As Integer
        Dim column As Integer
        Do
            press = Console.ReadKey(True).Key
            If press = ConsoleKey.A Then
                If Console.CursorLeft <> 1 Then
                    Console.SetCursorPosition(Console.CursorLeft - 4,
Console.CursorTop)
                End If
            ElseIf press = ConsoleKey.W Then
                If Console.CursorTop <> 2 Then
                    Console.SetCursorPosition(Console.CursorLeft,
Console.CursorTop - 2)
                End If
            ElseIf press = ConsoleKey.D Then
                If Console.CursorLeft <> 9 Then
                    Console.SetCursorPosition(Console.CursorLeft + 4,
Console.CursorTop)
                End If
            ElseIf press = ConsoleKey.S Then
                If Console.CursorTop <> 6 Then
                    Console.SetCursorPosition(Console.CursorLeft,
Console.CursorTop + 2)
                End If
            End If
        Loop Until press = ConsoleKey.Enter
        Select Case Console.CursorLeft
            Case 1
                column = 0
            Case 5
                column = 1
            Case 9
                column = 2
        End Select
        Select Case Console.CursorTop
            Case 2
                row = 0
            Case 4
                row = 1
            Case 6
                row = 2
```

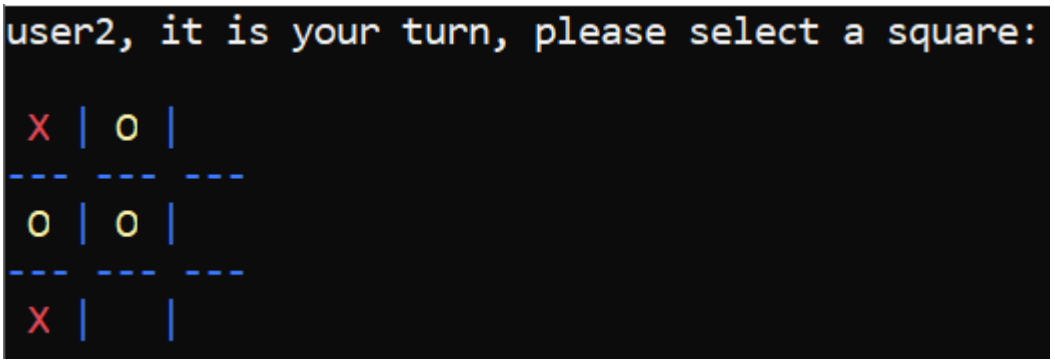
```
        End Select
        If Board(row, column) <> " " Then
            Console.Clear()
            Console.ForegroundColor = ConsoleColor.White
            Console.WriteLine("Space is already taken, please try
again:")
            Console.WriteLine(" ")
            DisplayBoard()
            TraverseBoard(Current)
        Else
            Board(row, column) = Current.token
            Console.Write(Board(row, column))
        End If
    End Sub

    Sub DisplayBoard()
        Console.ForegroundColor = ConsoleColor.Blue
        For i = 0 To 2
            For j = 0 To 2
                Console.Write(" ")
                If Board(i, j) = "X" Then
                    Console.ForegroundColor = ConsoleColor.Red
                ElseIf Board(i, j) = "O" Then
                    Console.ForegroundColor = ConsoleColor.Yellow
                End If
                Console.Write(Board(i, j))
                Console.ForegroundColor = ConsoleColor.Blue
                Console.Write(" ")
                If j <> 2 Then
                    Console.Write("|")
                End If
            Next
            If i <> 2 Then
                Console.WriteLine("
--- --- ---")
            End If
        Next
        Console.ForegroundColor = ConsoleColor.White
    End Sub

    Function isWon() As Boolean
        If Board(0, 0) = Board(0, 1) And Board(0, 1) = Board(0, 2) And
Board(0, 0) <> " " Then
            Return True
        ElseIf Board(1, 0) = Board(1, 1) And Board(1, 1) = Board(1, 2)
And Board(1, 0) <> " " Then
            Return True
```

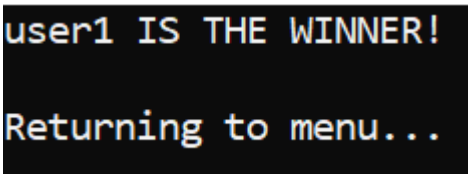
```
        ElseIf Board(2, 0) = Board(2, 1) And Board(2, 1) = Board(2, 2)
And Board(2, 0) <> " " Then
            Return True
        ElseIf Board(0, 0) = Board(1, 0) And Board(1, 0) = Board(2, 0)
And Board(0, 0) <> " " Then
            Return True
        ElseIf Board(0, 1) = Board(1, 1) And Board(1, 1) = Board(2, 1)
And Board(0, 1) <> " " Then
            Return True
        ElseIf Board(0, 2) = Board(1, 2) And Board(1, 2) = Board(2, 2)
And Board(0, 2) <> " " Then
            Return True
        ElseIf Board(0, 0) = Board(1, 1) And Board(1, 1) = Board(2, 2)
And Board(0, 0) <> " " Then
            Return True
        ElseIf Board(0, 2) = Board(1, 1) And Board(1, 1) = Board(2, 0)
And Board(0, 2) <> " " Then
            Return True
        End If
        Return False
    End Function

End Module



Public Class Player
    Public token As String
    Public name As String
    Public won As Boolean

    Public Sub New(side As String, user As String)
        token = side
        name = user
        won = False
    End Sub
End Class
```

Screenshots:









Overall Notes/Thoughts:

Firstly, while the client - server prototype was just to correctly set-up a host server and have a client be capable of sending information to it, I am happy to report that it was completely successful, with a test user (Interviewee 1) being capable of sending messages to me over the internet, between two different machines on two different networks.

Secondly, although the game example is a much more simplified and basic version of connect 4 (TicTacToe), I am happy with how it turned out and it

allowed me to experiment and improve upon the potential user interface and also how it functions.

However, during my prototyping I did also attempt a prototype of the AI, which while progress was made, no considerable program was completed and so for fear of time management I refrained from finalising a prototype. This could present potential problems in the future, as while it is a part of the program which was desired, answers from my first interview suggest it shouldn't be the absolute focus of the project. I will ask questions in the second interview to confirm this as well as get some feedback on how I should proceed if the AI portion becomes infeasible.

1.5.2 Second interview

The interview was conducted over a live discord call. The responses below are the transcribed answers to my questions from two separate interviewees.

Q1: Previously, you stated that the most important mode for you would be the multiplayer function. Hypothetically, if a mode of the program wouldn't be possible due to time constraints, would you like this to be the main focus?

Interviewee 1: Hypothetically speaking, then yes the multiplayer mode should be the main focus which should definitely be present in the project, but ideally the game would have more available modes then just the multiplayer.
Interviewee 2: Definitely, I think the multiplayer is the most important part of the program. For me it is the biggest selling point of this program.

Q2: Within connect 4 it is possible for two players to tie, where neither manage to achieve 4 in a row. Is it important to you how this impacts scoring, if at all. (For example would both players score a point, would neither, would it have any impact on a leaderboard score for them, etc):

Interviewee 1: Honestly i think in this scenario it would be easier and more convenient for neither player to score anything, and if necessary for the match to be replayed until a winner is determined.

Interviewee 2: If there is some sort of ranking/rating system, aside from just overall wins, it would be interesting if when a weaker player managed to draw against a stronger player, then there would be some increase in rating for the

weaker player, and some decrease in rating for the stronger player. This is a commonly used system by other games with a leaderboard system and I find it the most competitive approach.

Q3: In terms of the User Interface, would you say appearance or ease of use is more important? Is there anything about the User Interface you would like specifically incorporated?

Interviewee 1: Ease of use would be the more important of the two, connect 4 is a simple game for a wide and varied audience so ease of access is by far more important. As for anything specific as long as it is simple and effective I think that is all that's important.

Interviewee 2: Personally, ease of use matters more, but the UI should also be at least some-what appealing to the user, or else it could be deterring for them. I also think just making sure the UI isn't overly cluttered is important, and this would benefit both appearance and ease of use too.

Q4: For the networking of the multiplayer to function, one of our players would need to port-forward their machine to allow for hosting, which at a basic level means allowing the client connection to reach their machine by allowing the port through any router or computer firewalls. Do you see any problems with this?

Interviewee 1: While it does sound a bit intimidating, if it's a crucial part of the multiplayer then I don't see an issue with it, as long as this is the best way you've found to enable the multiplayer mode and it isn't overly hard to do.

Interviewee 2: I'm not entirely sure of what that would consist of but i don't really see a problem as long as someone who might not be super familiar with computers would be able to do it.

Q5: Do you have any comments about the game example prototype, like any significant issues you have with it or anything you definitely are happy with?

Interviewee 1: I don't have too much to say on it, as it is just a basic version of how the gameplay should work, but it is looking good so far and when it is at a more upgraded version i think it will be very good.

<u>Interviewee 2:</u> I really liked how actually playing the game went, it was quick and simple, which is just how it should be. Obviously the actual program will be a bit more advanced, but making sure we see the game playing out well across all game modes is the most important thing I can think of in regards to the game as a whole.

<u>Evaluation after interview:</u>

As was discovered before, it is definite that the multiplayer should be the mode which is definitely available in the finished program. While I still very much plan to still have an AI mode by the end, if it needs to be simpler than originally planned to accommodate the multiplayer and time restraints then so be it. Overall it seems that keeping the game simple but appealing is the main outcome to aim for, which is good because that is what i was planning to go for from the beginning, as it is one of the charms of a board game such as connect 4. Providing instructions on port-forwarding is going to be interesting, while completely manageable, different internet service providers also have different methods at approaching port-forwarding, but hopefully by making instructions as simple as possible it shouldn't become an issue.

1.6 Objectives

> Design a digital version of Connect 4 with multiplayer and AI capabilities, playable with one or two players.

1. Design Connect 4 Board
   1.1. Board of scale (7 x 6) (Columns x Rows).
   1.2. Board should be coloured Blue.
   1.3. Board should have spaces to fit counters, which also correlates to the dimensions (7 columns of spaces, 6 rows of spaces).
2. Design Playing Counters
   2.1. Counters of two colours (Red and Yellow).
   2.2. Counters should have dimensions that match board spaces, mustn’t exceed or be shorter than.
3. Connect 4 Rules
   3.1. Host of the lobby (Player 1) Always goes first.
   3.2. Players may choose their token colour and name.
   3.3. Counters can be placed into the board.

3.3.1 Counters can be placed down one of the 7 columns.

3.3.2 Counters must be placed at the lowest point of the column, which is either the bottom of the board or on top of any counters already within the column.

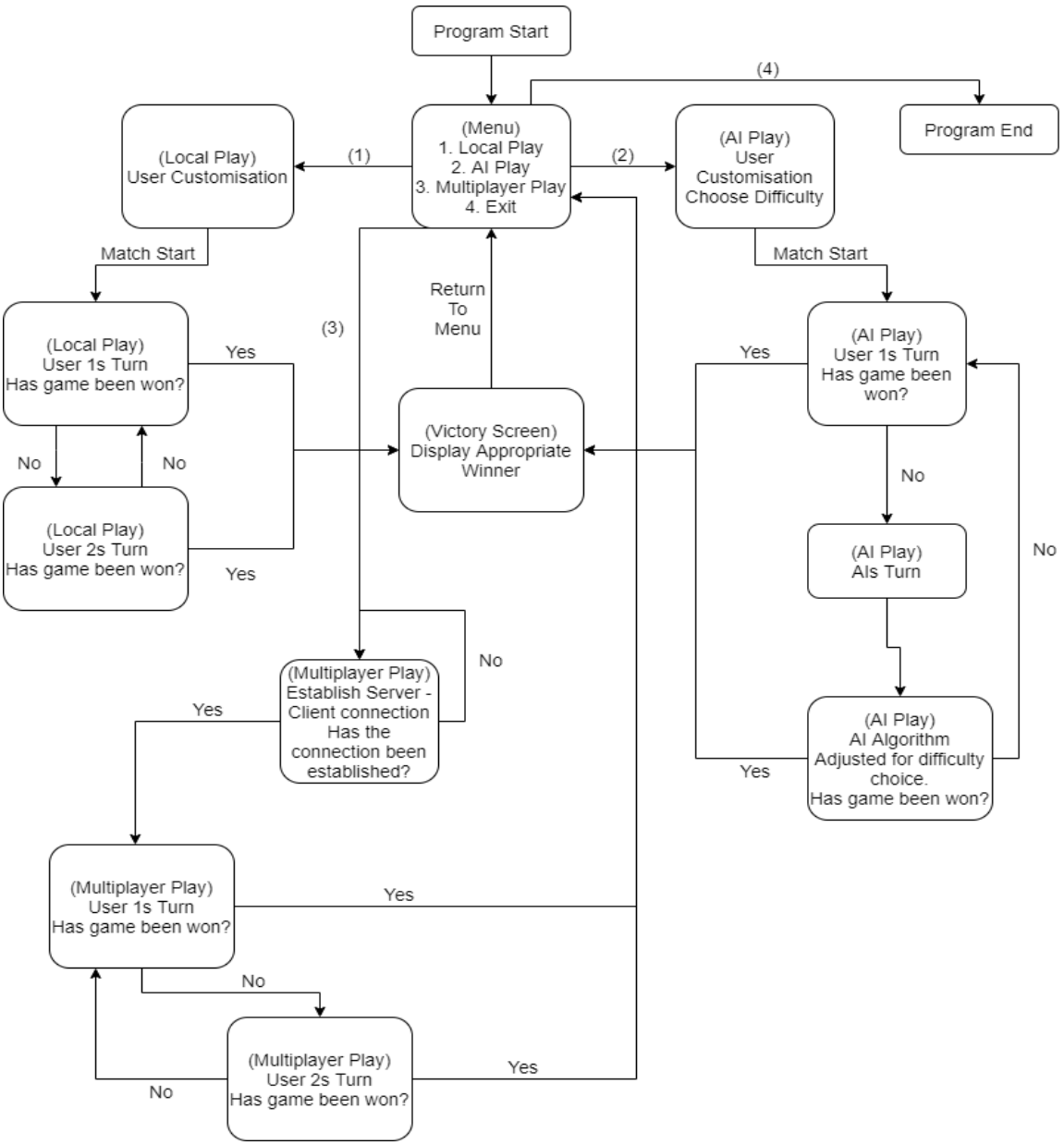
3.4. Players will alternate turns, placing one counter each time.

3.5. The game must end when either player achieves 4 of their counters in a row (This can be of any orientation).

4. Networking Capabilities
   4.1. Have the client enter an IP address of the host computer.
   4.2. Establish connections between client and host.
   4.3. Allow users to pick preferences.
   4.4. Ensure users can control and take their turns one after the other.
5. AI
   5.1. AI programmed with rules of connect 4.
   5.2. Apply Algorithms (MinMax and Alpha-Beta Pruning).
   5.3. Allows the user to play against AI.
6. LeaderBoard
   6.1. Create a leaderboard, accessible from the menu.
   6.2. Have leaderboard update upon opening.
   6.3. Add/Update a user depending on if they've already won or not.
7. Local Play
   7.1. User(s) able to play against each other from the same computer.
   7.2. Rematch available after each game.

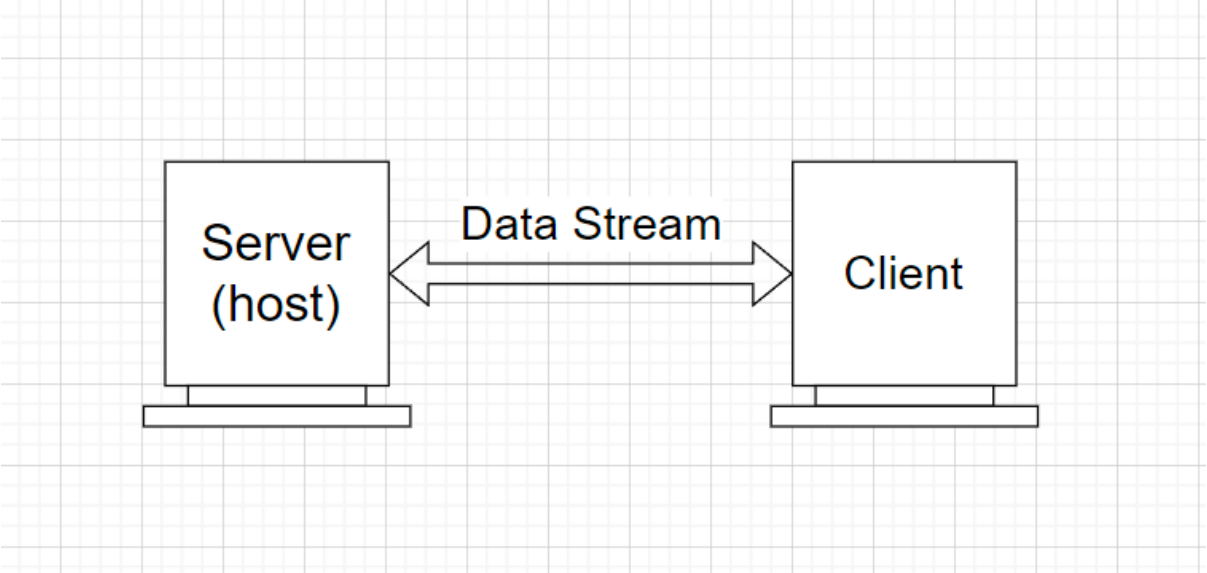
1.7 Modelling

Below is a flowchart of how I want the game menu/systems to be traversed by the user, it should make the layout of the game easier for me to keep a track of.



Additionally, my project will take advantage of a simple client-server model for the multiplayer part of my project. In the case of my project, this will always reflect a one-to one relationship between the server and the client. This is due to the fact that Connect 4 is a two player game and that one of the two player's will be hosting the server from their computer for my program.It is completely unnecessary for the server to be able to communicate with multiple clients at once instead of just one. A client will request a connection with the server (the host), using their public IP address, which will be accepted by the awaiting server. Both parties can then communicate to each other over a data stream, with which they can send and receive information from each other.

Below is a heavily abstracted diagram demonstrating the simplicity of my client server model:

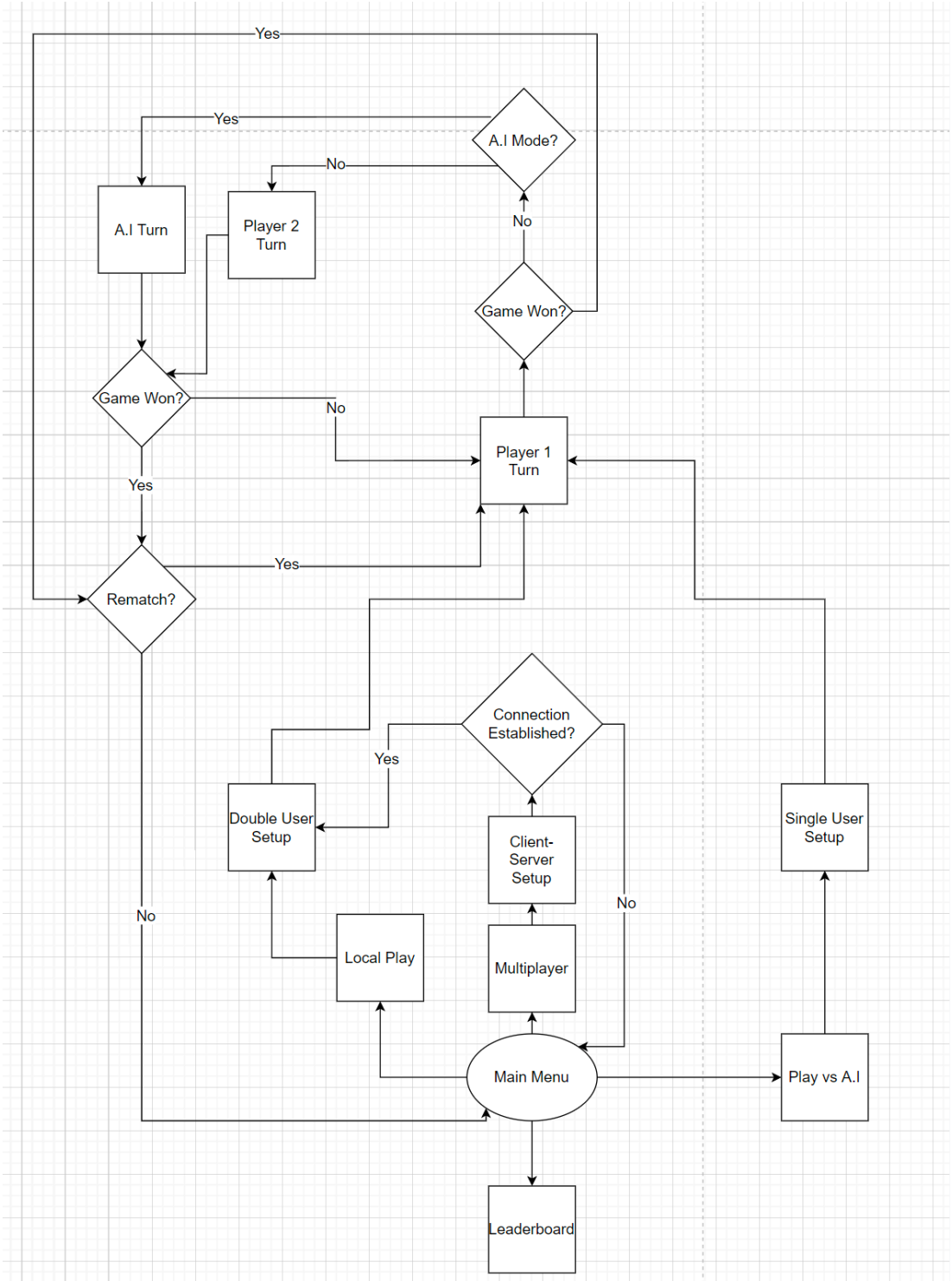


As can be seen above the model works very similarly to a “Peer to Peer” network, however it is only the host here which ever starts a server, the client is simply connected to this server. Both systems can exchange information between each other through the use of a TCP connection. This is what will enable the multiplayer feature of my project to perform appropriately.

2 Design

2.1 High Level Overview

System Flowchart of program functionality for the user:



The objective of my project is to be a simple game that will give people the ability to play against their friends (be it in person or online) or practice by themselves against an A.I. The key features of my project are the online aspect, consisting of a client-server relationship, an A.I mode utilising algorithms to determine the moves made, and a sorted leaderboard for tracking individual user progress. As my program works as a game with effectively different modes, they act independently only until the game begins, at which point the actual interaction from the user will feel identical regardless of the mode. This has the exception of the Leaderboard, which is affected by every mode; whenever a user wins, it is updated.

From the diagram above, you can see the structure of the program, according to the user's interaction with it. It is heavily abstracted and simpler than the one seen in modelling (Section 1.7). It doesn't portray the complicated processes working behind the scenes or the smaller less significant details such as a user setting their name with the user setup. Additionally, it doesn't represent any code which runs without any user input/acknowledgement, such as the sorting when the leaderboard option is chosen at the menu. At a basic level, the program will always follow these general stages:

1) Gamemode selection (Main Menu)
2) User setup
3) Player 1 turn
4) Player 2 turn
5) Repeat steps 3-4 until the game is won.
6) Consider rematch
7) If yes, return to step 3
8) If no, return to step 1

This of course does not consider the leaderboard option, or alternatively an exit option which wasn't included in the flowchart as it isn't relevant to the program. Due to the simplicity of the game (connect 4) as well as the system model which the user follows, the main content of the program can be contained within these steps. This also makes it simpler for me to conceptualise during coding and when making modifications to separate game modes.

Furthermore, the rules of connect 4 need to be considered within the coding of my program, as these should guide how i allow the user to interact with the

game as well as how the A.I scores and selects it's best move. The rules/setup of connect 4 are relatively simple as bulleted below:

- There is a board of dimensions 7x6 (Column x Row).
- Player's will decide between one of two coloured tokens.
- These tokens can be placed in any of 7 columns, but they must be placed at the lowest available slot of each column.
- Player's alternate turns where the place a single token into a column
- This repeats until a player has 4 tokens in a series in any orientation, horizontally, vertically or diagonally.
- When a player achieves this, they win, otherwise the game progresses until every slot is occupied by a token, in which case the game ends as a draw.

This is simple enough to conceptualise and won't be hard to follow/implement in my program. The main things to consider will be the alternating turns, the fact that all tokens fall into the lowest available slot and the win condition.

2.2 Choice of programming language/libraries used

The language my program will be written in is VB.NET (Visual Basic). VB.net is a high level .NET language which supports OOP (Object oriented programming), which is relatively simple to understand and utilise. It additionally has simple multithreading functionality which I will be making use of in my program. However, the main reason I chose this as my programming language is because it is the language with which i have the most experience. It is the language I have spent effectively all of my time at college working with and so subsequently most of my developed coding knowledge is within this language. This means that i will be able to spend more time focusing on the actual content of my program rather than having to learn or adapt what i know to a language im less familiar with.

I will be using a few libraries in my program, as these will be particularly useful for the network portion of my program, as well as with the leaderboard as my current plan is to incorporate a text file to store the data for the leaderboard.

The first library I will need is “System.IO”. This is a very commonly used library, as it is crucial within .NET when interacting with files and data streams, both of which are relevant to separate areas of my project. When it comes to dealing with a text file, this library provides the ability to very easily read and write to the file, by using streams provided such as “StreamReader” and “StreamReader”. Using the appropriate file address, these can be used to easily store the contents of a text file to a string, string array, list etc within my program. This will be useful for updating/sorting the leaderboard while the program is running. Additionally, these applications of streams are useful for networking as well. This is because the communication of data between a host and client is done through the use of data streams, and I will be able to send/receive data on this datastream using this library.

The second/third libraries I will be using are “System.Net” and “System.Net.Sockets”. The first is a library I am simply using to store an IP address within my program, which I can then use to setup a “TcpListener” class. This class, as well as “TcpClient” are classes provided by the “System.Net.Sockets” Library. These are essential to my program as my client-server system will rely on a TCP connection and I need these classes to provide the foundations for my client and server connections. The listener class starts the server and listens for any client connections which are incoming onto the host IP address and designated port. The client class provides the ability to attempt to connect to a server by using the host address and relevant port number.

The final library I plan to implement is “System.Threading”. This gives my program access to simple, efficient multi threading which will be useful for me when it comes to networking, once again. This is because my program may need to be able to do other things while it is waiting for a potential incoming connection, and this will be possible through multithreading. For example, reading any data coming in through the readstream while attempting to send data to the stream.

2.3 Design Techniques

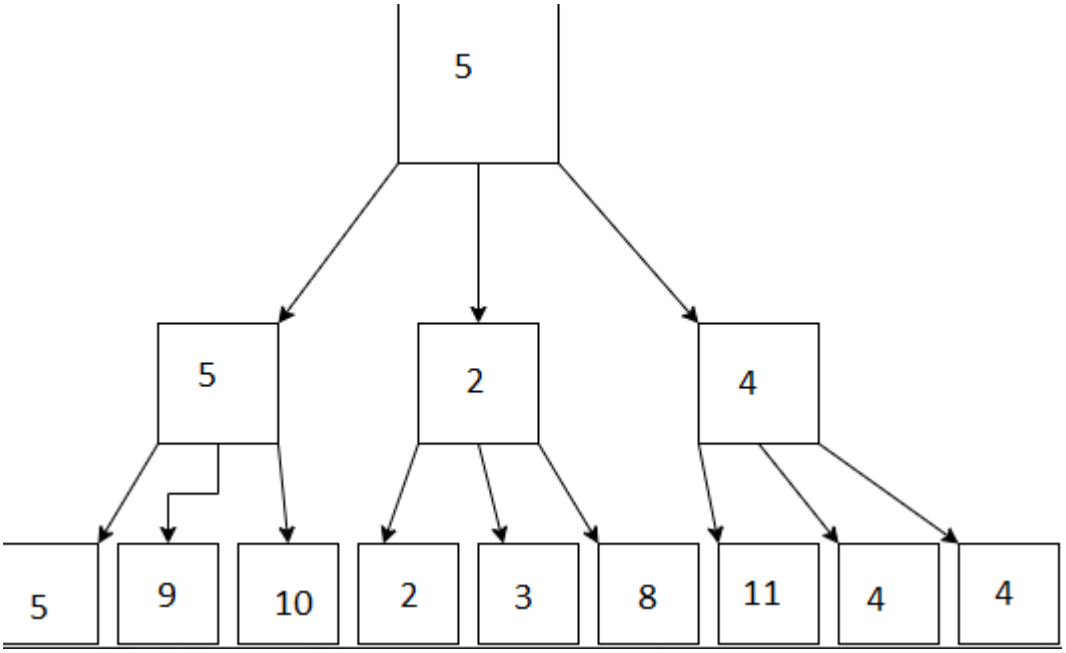
2.3.1 A.I Mode

Algorithm Design

The two fundamental algorithms I will be implementing for the A.I section of my project are the MinMax algorithm and an optimisation which will work as an improvement to the MinMax algorithm, Alpha-beta pruning. These methods combined can be found frequently within two-player game A.I's such as Chess or noughts and crosses.

MinMax

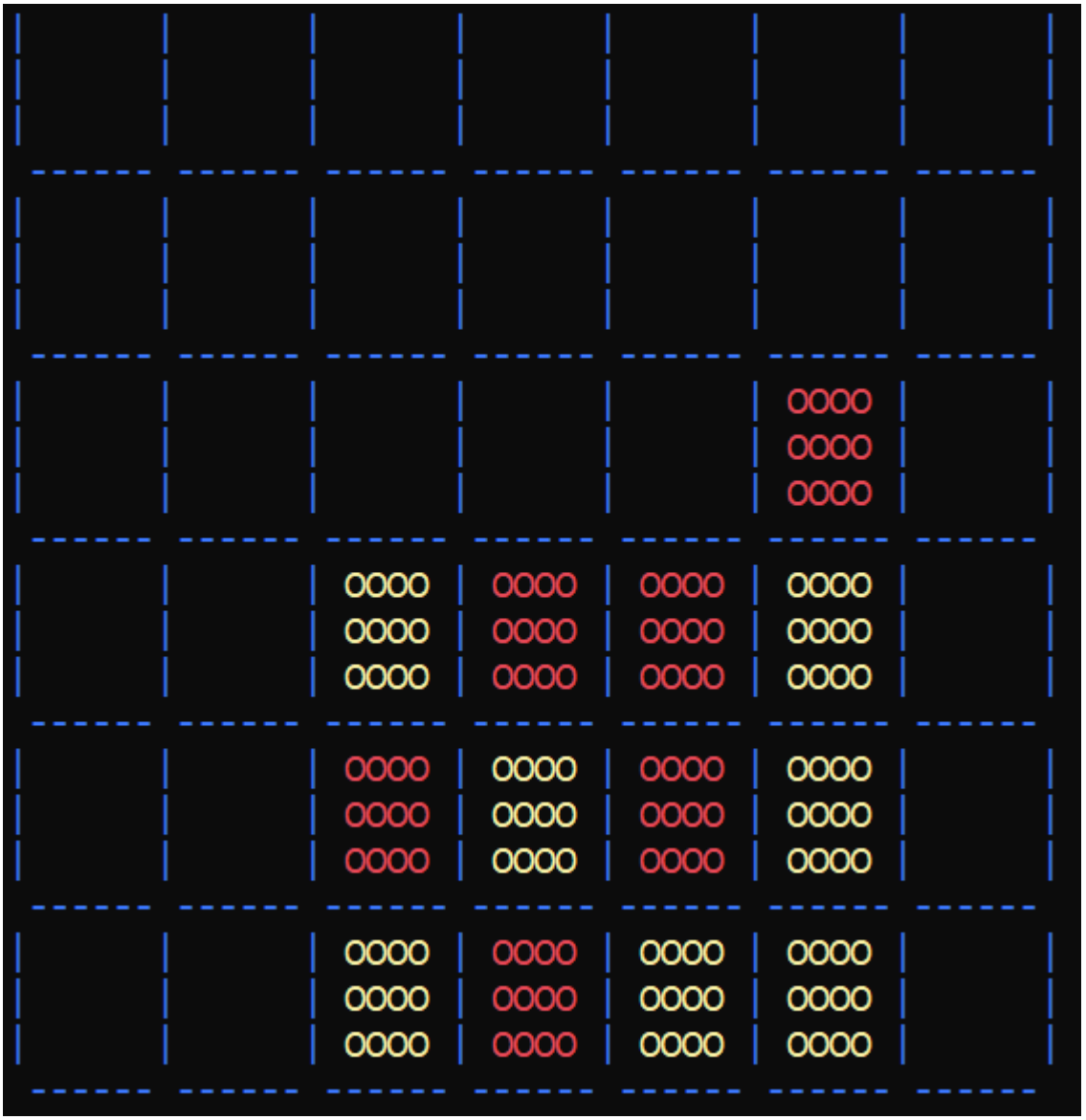
MinMax is a very common, recursive game algorithm which uses informative decision making to identify the most optimal move for a given game state. In MinMix, each player within the two player game acts as either a maximiser or a minimiser, where the maximiser is trying to increase the score as much as possible while the minimiser tries to reduce the score as much as possible. So within any given game, if the maximiser is seemingly winning, the current score would be positive, whereas if the minimiser was winning the score would be negative. So in Connect 4 how can you numerically evaluate the score of each board state? Well, it is less about the tokens currently shown on the board, and more about what will eventually happen x moves down the line. The algorithm has a specified depth, in this case x, which continues the consideration of possible moves past just the next one. For each potential board state after the next move, every possible move from those 7 board states will then be considered and so on. The scoring of each move can then be based on which moves have the highest "chance" of reaching a won position, assuming the opponent, the minimiser, plays perfectly. This will optimise the A.I's ability to win as it will always be skewing the game towards an outcome which favours it. Below is a chart which demonstrates this method.



This chart is a very simple demonstration of how MinMax works with real values. Here we have a representation of this algorithm with a depth of 2, and a potential move set of 3. If we assume it is the maximisers turn to make a move we can follow the method through each stage. Firstly, no decision making will be made until every single outcome available at the specified depth has been considered and scored (as represented by the bottom row of the flowchart). We can see 9 different possible outcomes each of which has been scored based on how favourable it is, ranging from 2 to 11. The “best” move available here is obviously the one with the highest score, which in this case would be 11. But, the maximiser must always be under the assumption that the opposing minimiser will be playing perfectly. Therefore, if the maximiser did make the move which could potentially result in a board state with score 11, it would actually end up with a score of 4, as the minimiser would always act to reduce the score as much as possible. By understanding this line of reasoning, we can actually see that taking the move represented by the leftmost node, is actually more beneficial. This is because while taking into account the actions of the minimiser, the highest score we could possibly reach by the 2nd move is 5, as it is the most the minimiser would be able to reduce the score to.

As a demonstration of how this can be applied within connect 4, we can take a board state and evaluate how the algorithm would consider each potential move:

1 2 3 4 5 6 7



Human Player 1 : Yellow A.I Player 2 : Red

In the above example, this is the current state of the game and it is red's turn to make a move. Red in this case is the A.I employing the MinMax algorithm and so is going to consider each move available out of the 7 possible. If we assume that the scoring will be based on how good the next move is at putting us in a winning position (maximising our score) then we can give a brief evaluation of the possible moves.

Both columns 1 and 7 are moves which have little impact on how the game is currently playing out. All tokens so far have been placed in columns 3-6, and placing a token in either 1 or 7 will not connect with any of red's other tokens, essentially providing no benefit for red, so neither of these moves would be ideal. Placing a token in column 2 or 6 appear initially to be great moves. They open up the threat of 4 in a row which would lead to a win for red. However, if we look further ahead, we can see that after yellow places a token in column 5, red is left with little to no threat of victory from the tokens currently available, so either of these moves would be considered as just good but still not ideal. Columns 3 and 4 follow in similar fashion to 1 and 7, although they have the benefit of at least creating a 2 or 3 in a row for red, so have at least a little usefulness associated with them.

Nonetheless, the algorithm would eventually determine that placing a token in column 5 would be the highest rated move. This is because this move provides 3 separate ways of achieving 4 in a row, hence a win and it is impossible for the opposition to block all of these ways in one move. Subsequently this move would be scored the highest and played by the A.I.

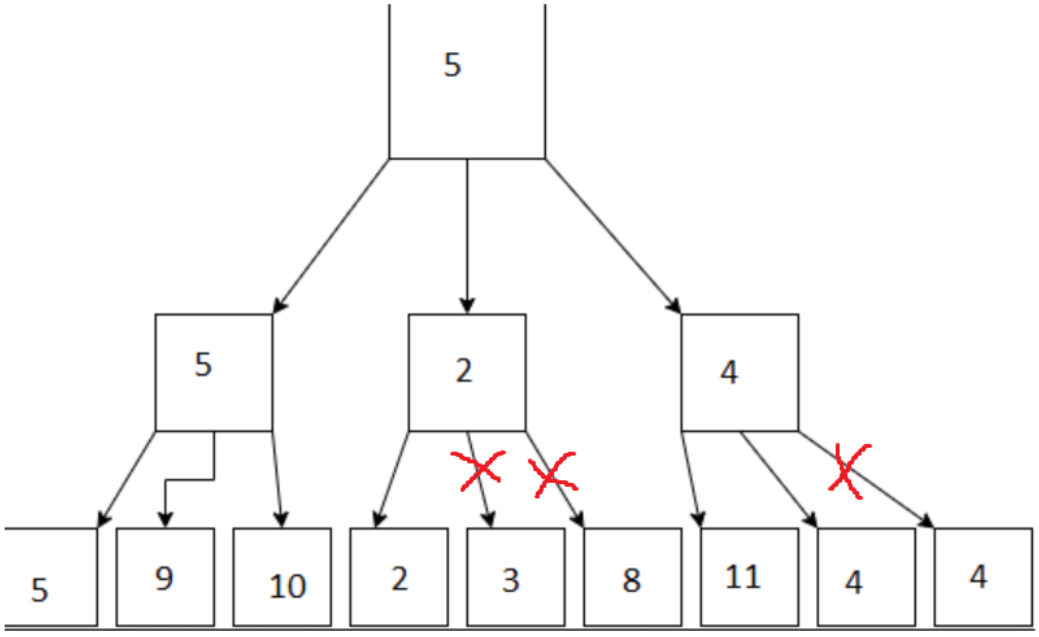
Obviously the algorithm isn't making human consideration of these moves, it is using a predetermined scoring format which would take into account some of the things described by treating them as different board states. For example, achieving 2 in a row may increase the score by 1, or 3 in a row may increase the score for that move by 5 as it opens up a winning opportunity. Additionally, preventing an opponent from achieving a 4 in a row in the next move may be worth 50 points so that this move is chosen over others as it prevents losing. By making numerical observations of each move and it's consequences, the algorithm can identify the best move for each position it has to, irrespective of the variety of potential different board states in Connect 4.

Alpha-beta pruning

Alpha-beta pruning is an optimisation technique which can be applied to MinMax to reduce computation time by disregarding options which will never be chosen as there already exists an option(s) which will always be better. This is an excellent optimisation strategy because the amount of moves the MinMax algorithm will need to consider grows exponentially as the depth increases. For example, at depth 1, we are looking at 7^1 number of possible

moves, so 7. At depth 2, we are looking at 7^2 number of possible moves, so 49. The exponent of 7 increases by 1 as the depth does the same, meaning that by only depth 6, we would be looking at 117649 possible different moves that the algorithm would need to individually consider.

This is where the method of Alpha-beta pruning becomes effective. Essentially, what this algorithm does is it disregards possible options from the search of the MinMax algorithm if it is already certain that those options would never be chosen. In terms of a search tree, it ignores branches of the tree which would never be traversed, without caring for the value of the node at the end of the branch. We can use the chart previously demonstrated in the MinMax part of this section to demonstrate how alpha beta pruning would help to effectively reduce the number of considerations made to the same tree.



In the above chart, branches with a red X through them are one we didn't even need to consider due to the implementation of alpha beta pruning. For the first branch to the left, all 3 sub-branches need to be considered, as we don't have any values which can be considered as the minimum possible score yet. After considering these, we know that the lowest score achievable is 5, so this is what the minimiser would choose, assuming we made this move. As we move to the middle branch, we can see that the left-most sub-branch, the one considered first, provides the algorithm with a score of 2. This means that, without knowing about or even of any other sub-branches, we know that the score the minimiser would choose if the above move was

made by the maximiser is equal to or less than 2. Therefore, we know the maximiser would never make the move represented by the centre branch, as a score of 5 is already reachable through the move represented by the left branch. A similar line of thinking can be said for the right most branch. As soon as a score of 4 is found, the remaining branches can be immediately disregarded, as we know that a score of 5 is already possible elsewhere.

The Alpha-beta algorithm will always reach the same result as the MinMax, after all it is just an optimisation, it doesn't affect the core functionality of the original algorithm. What it does do, however, is reduce the computational time taken for the algorithm to reach the end result, by omitting many options that MinMax would waste time and processing power to consider. Even in the example given, it reduced the number of considerations by 33%, which is a massive improvement. If it did the same on a much larger sample size, this would substantially reduce the time taken and processing power expended.

2.3.2 Multiplayer Mode

Client-Server Model

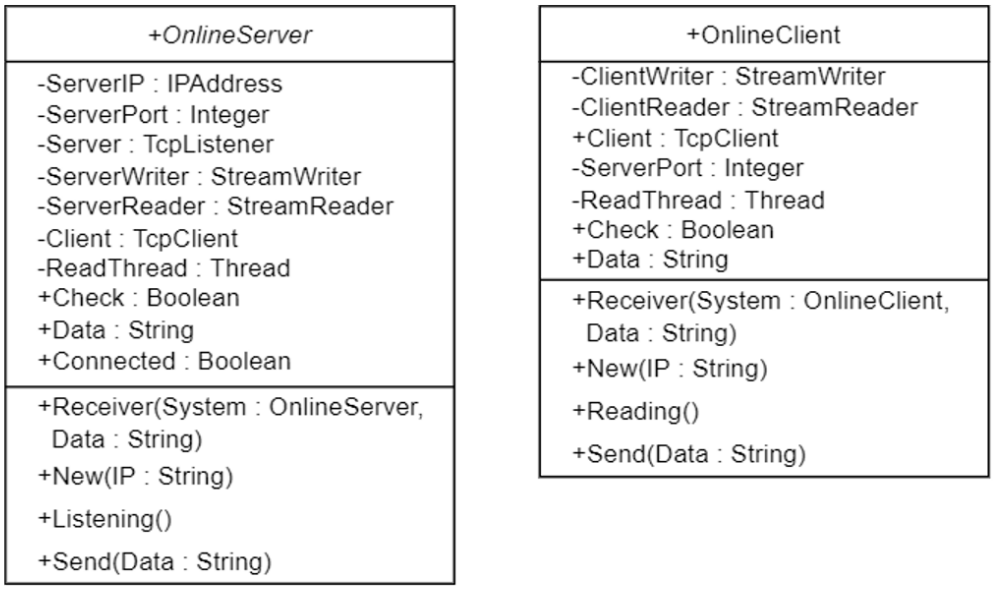
When it comes to the networking part of my project, my program makes use of a simple client-server model to communicate information such as the move made and each user's preferences between 2 different systems. This could be between two systems on the same network or two systems which exist within completely different networks. Within my program there are two classes, which control the server side of the model and the client side of the model respectively. These are "OnlineServer" and "OnlineClient", both of which take advantage of the previously mentioned libraries (Section 2.2).

Firstly, "OnlineServer", which is the class which controls the networked part of my program for the host/server. It takes an IP address as an argument and instantiates an object of class type TcpListener which itself belongs to the library "System.Net.Sockets". This object listens for any incoming connection attempts on the specified IP address and port. I have designated the port to always be "1234" as this is a port outside of the range of well-known port numbers and is a port which is highly unlikely to already be in use on a home pc. The instantiation of "OnlineServer" also starts a thread which is continually attempting to detect any pending connection requests to the server, through the use of a subroutine called "Listening". Threading will be expanded on in

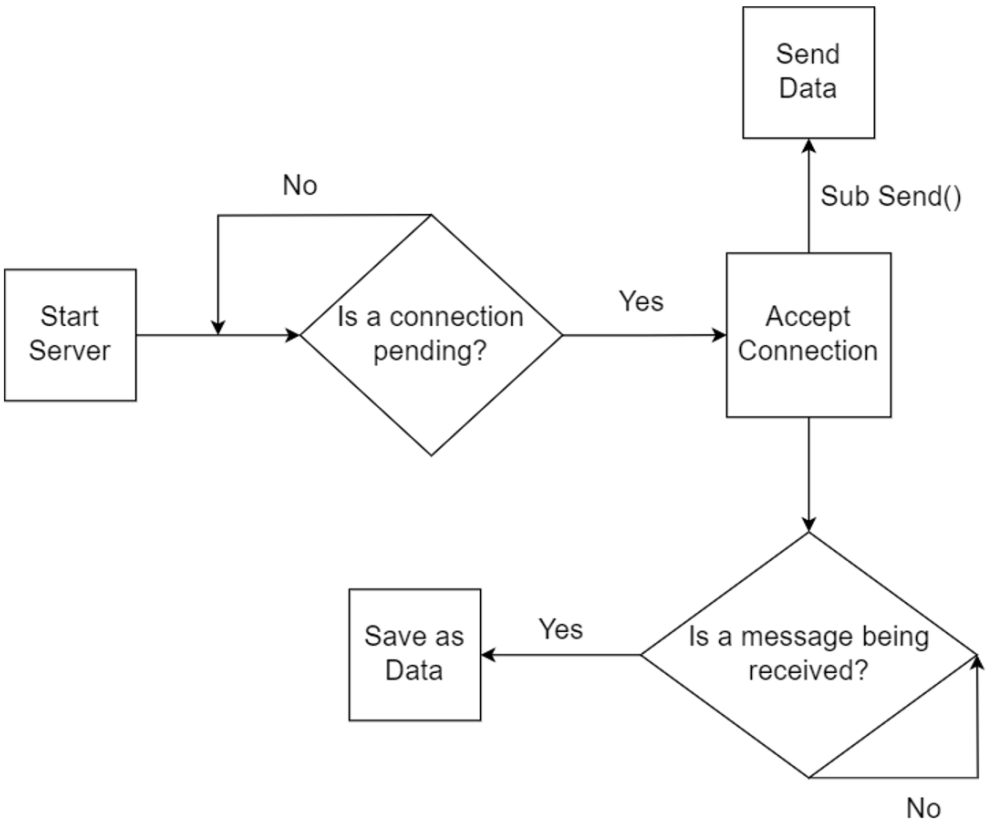
the next segment, Multithreading Architecture. When the TcpListener class detects an incoming connection request, it is instructed to accept this connection and instantiate a new object as a “TcpClient”. This is another very useful class which provides the program with client connections for our TCP network. It then initialises a StreamReader and StreamWriter, to both send and receive information from the data stream now established between the client and server. Finally it indicates that a connection has been established and begins attempting to read any data sent through the stream, with a time buffer of 100 milliseconds to prevent any errors or excess processing expenditure. Finally, a subroutine called “Send” is used to write data to the stream so that it can be read by the client system.

“OnlineClient” is the class which controls the client-side networking. Similarly to the server class, this class also takes only an IP address as an argument which is then used to instantiate a TcpClient using the host’s IP address and the port number 1234. This class also initialises both a StreamReader and StreamWriter for reading and writing to the data stream and likewise starts a thread for attempting to read any data from the stream, using the subroutine “Reading”. Subroutine “Send” is once again used to write data to the stream. The main difference between these two classes is that the client requires the server to be started to be instantiated and the subroutine “Reading” doesn’t await for any connections like “Listening” does.

Both classes can be represented below by boxes:



As we can see, and as previously described, both classes are similar but do have differences which are what define them individually. Their interaction is very simple, and is purely a read/write system, which is all that is necessary for my project. Below is an example of how this interaction would work in the context of my project.



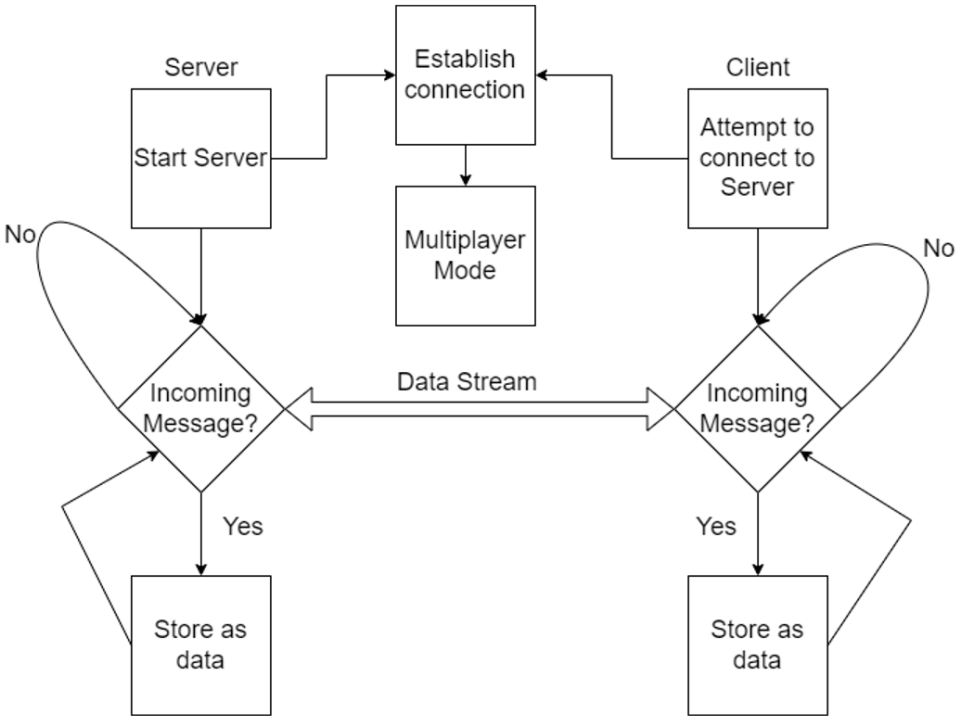
This is how the server class functions in terms of starting and sending/receiving data. The client class works very similarly, however it has no need to await a connection, as one will be established when the class is instantiated with the correct IP.

I use these classes, in conjunction with my Connect 4 game design, to communicate inputs between two different systems on different networks. When player 1 is making a move, player 2 can see it happen as it is happening, and won't be able to make their move until player 1 has finished their turn. Additionally, when the users are choosing their preferences, the client's system is aware that the host is currently choosing their preferences and so must wait until they are done before choosing their own. When it comes to the rematch request at the end of each match, both players make their choice, and each player's choice is communicated to the other's system.

If both are in agreement, another match begins, however if they disagree, then both are returned to the menu.

Multithreading Architecture

The classes which I've setup for the multiplayer mode both take advantage of multithreading so that they function appropriately. For an online multiplayer mode to work suitably, both the client and server must be ready to receive information from each other at any given period of time, while the main thread of the project is running. This is where multithreading has been applied. By initialising a new thread on both user's systems, whose only job is to constantly be attempting to read data from the stream, we can ensure that both receive the information required at the correct time to progress through a multiplayer game. In both "OnlineClient" and the "OnlineServer" classes, this thread is labelled as "ReadThread", although each functions slightly differently. Within both classes, this thread contains code that attempts to read data from the stream at a constant interval of 100ms, which works as a time buffer. However, within "OnlineServer" there is an additional clause within this thread. If there is a pending incoming connection to the server, this thread first instantiates an object of class TcpClient, initialises the subsequent read and write streams and then begins the loop of attempting to read data from the stream. A worked example of this, with context to my program, can be seen below:



As can be seen from the chart above, once both client and server have established a connection to each other, the threads are started to read from the data stream. This is completely irrespective of what is going on in the main thread and instead we take advantage of these two threads which are running simultaneously with the main program to communicate information about what each player is doing. For example, whenever it is player 1's turn, any input he makes will be sent to the data stream and subsequently read by player 2's ReadThread. The data which is read is subsequently stored in a public variable appropriately named "Data", within the respective class, so it can be accessed by the main program. In addition to this, a variable called "Check", which is stored as a boolean, has its value changed to true whenever a new piece of information is stored to data, which is a useful way to allow our main thread to be aware of whenever new data has been received. This process works for both client and server.

2.3.3 Leaderboard

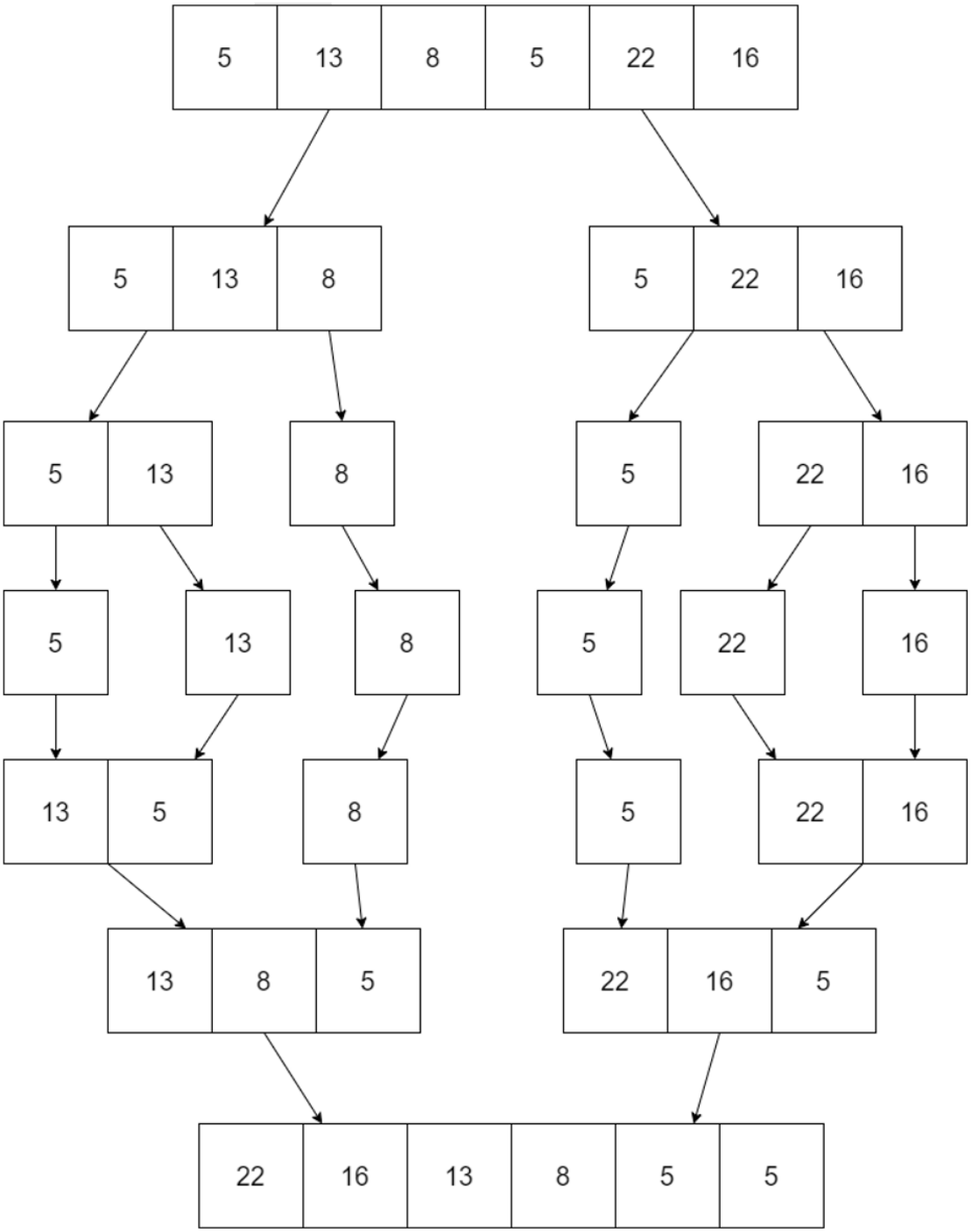
Merge Sort Algorithm

My leaderboard runs a merge sort algorithm whenever it is opened, as a way to sort the current users which exist within the leaderboard from most wins to least wins, with each player being assigned a number based on how high up the leaderboard they are. 1 is the most wins, 2 is the second most wins etc. This was a necessary implementation as an unsorted leaderboard doesn't fully achieve its purpose of giving players an estimate of how good they are compared to other players.

I was able to fully implement my merge sort algorithm recursively, and the entirety of the sorting exists between two functions. The first function, "MergeSort", takes as arguments an array of numbers, the values which are to be sorted, and also a pointer to indicate the midpoint of the number of values within the array. In the case of an array with an odd number of elements, the midpoint is rounded up to avoid errors. This rounding has no impact on the effectiveness or functionality of the sorting algorithm. This function then splits the inputted array at the midpoint, creating two new arrays, each with half of the elements in, or potentially one having one more element than the other. This process recurs until it splits an array consisting of two elements into two separate arrays which each contain only one element. At this point, the "Merge" function is called. This function takes the two arrays containing one

element each, and merges them in the intended order (highest to lowest etc), and returns the new sorted array of two elements. These two functions work in tandem, slowly reconstructing the original array of values into its sorted form, by merging all the split elements and sorting them in the process.

To demonstrate more effectively how this sorting algorithm works in code, below is a diagram demonstrating how the algorithm behaves, and a step by step of how the program follows the sort:



Merge sort algorithm steps on above example:

- An integer array containing the values 5, 13, 8, 5, 22 and 16 is passed as an argument into the function mergesort. These values represent the number of wins for a group of 6 users. The midpoint value, in this case 3, is also passed as an argument into the function.
- Two new arrays are defined within mergesort, first() and second(), each of length 3. First() is populated with the first 3 values of the initial array, 5, 13 and 8, while second() is populated with the last 3 values of the initial array, 5, 22 and 16.
- Two pointers are also stored, one for each of the new arrays, to represent the midpoints. In this case the value of both pointers is 2, as both arrays have the same length.
- These two arrays are then each passed into mergesort again with their respective pointers, which results in now 4 arrays of numbers, 2 of length 2 and 2 of length 1. The arrays of length 2 contain the values 13, 5 and 22, 16 correspondingly, with the arrays of length 1 respectively containing 8 and 5. This demonstrates the recursiveness of the merge sort algorithm, as this step is repeated a further time until we have only arrays of length 1.
- The final time the merge sort algorithm recurs, we are left with 6 arrays, each containing one of the 6 original values from the original array.
- We now move onto the merging part of the algorithm, found within the “Merge” function. Here we begin to recombine the values while simultaneously arranging them in order, in this case from highest value to lowest value.
- In the first stage of the merge function, from the diagram above, we can see that only 4 of the 6 arrays are actually merged. This is because there was an uneven number of values in each array after the first split, so we have to first combine 4 of them together, and then combine the remaining two with the two new arrays.
- When 5 and 13 merge, 13 is the larger number and so this is the first value added to the new array, with 5 being added second. 8 and also the second 5, do not merge at this stage, as they were not split from any other values in the previous step. When 22 and 16 merge, 22 is the larger number and so is added to the array first, with 16 being added second. While this doesn’t actually change the order of the array from how it was as an array of length 2 before, the algorithm must still compare the values to identify which is larger.

- Next, the array containing 13 and 5 is merged with the array just containing 8. As 8 is less than 13, but greater than 5, the order of the new merged array is 13, 8, 5. When the array containing 22 and 16 is merged with the array containing 5, 5 is less than both values, so the new merged array is 22, 16, 5.
- The final step is to merge the two arrays [13, 8 ,5] and [22, 16, 5] to achieve the final sorted array. We can see that in this case, 22 is the largest number, followed by 16, 13, 8 and then the two 5s. Therefore the final array will be 22, 16, 13, 8, 5, 5 in that exact, sorted order.

This new array represents the sorted wins from 6 users, from highest to lowest. These player's can now be rewritten into the leaderboard, in this order, to correctly identify the rank of each player. This will run every time the leaderboard is opened, to ensure that whatever the user is able to view is always the intended order of the leaderboard.

File Structure

My leaderboard makes use of a .txt text file to store each user's name and the number of wins they have accrued. I did this so that even when the program is closed, the leaderboard is still completely saved to a user's pc. The way the information is very simple and an example of it can be seen below:

```
User1 18
User2 13
TheMan 8
NotTheMan 5
```

The name of a user and their number of wins are simply separated by a space, so that when the file is read, each line simply has to be split at the space in order for a name or win count to be altered. When changes are being made to the leaderboard, every line of the file is stored in an array, and accordingly split into names and wins. The number of wins for any user can then be incremented when they have won a game. The arrays are then recombined, and the entire file is rewritten using this array. This maintains the format of the file completely and will only seemingly change the intended portion, even though the entire file is being rewritten.

Similarly, when the leaderboard needs to be sorted, each line is read, the number of wins are sorted from highest to lowest and then each name and win count is rewritten to the file, except this time the order changes, in line with how the merge sort has indicated it should be.

2.3.4 Data Structure(s)

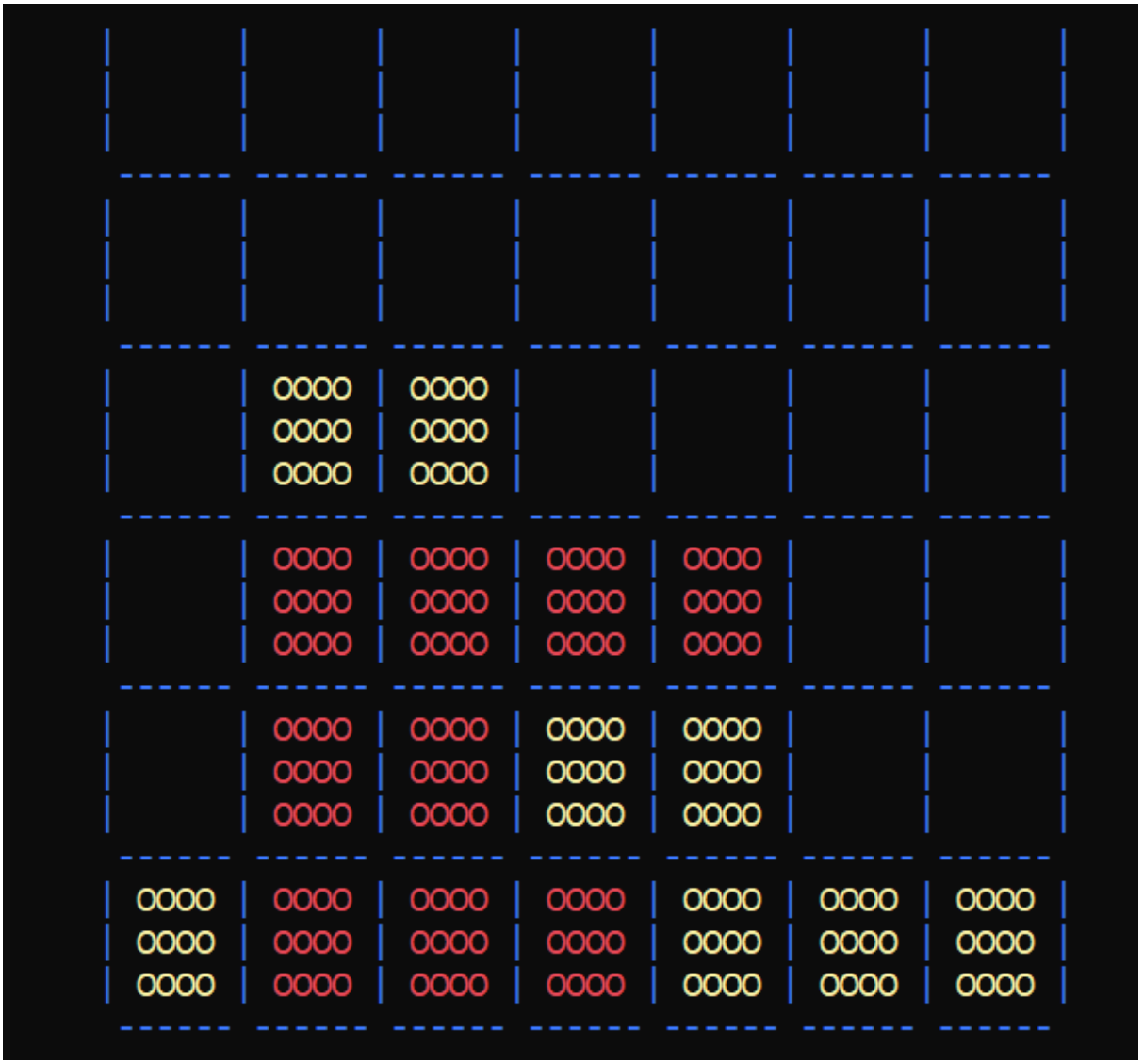
2D Array

The Connect 4 board (6 rows x 7 columns), which is displayed and affected by all playable game modes, is stored as a 2D string array. A 2D array is very useful for storing the Connect 4 board as it only has the 2 dimensions, rows and columns. The array, labelled Board, has specified dimensions (5, 6) as in VB.NET, arrays always index from 0 and so this is how I can achieve a conceptualised board of dimensions 6 x 7. The only values that each element of my array can ever actually be are either " ", "R" or "Y". This is representative of an empty space, a space occupied by a red token, or a space occupied by a yellow token. The board is initially populated by only " ", but as users make their desired moves, an element will change to either an "R" or "Y" for the corresponding token. The 2D array can be visualised like the diagram below, which can subsequently be directly compared with the connect 4 board state that the 2D array represents.

Columns

Rows

	0	1	2	3	4	5	6
0	" "	" "	" "	" "	" "	" "	" "
1	" "	" "	" "	" "	" "	" "	" "
2	" "	"Y"	"Y"	" "	" "	" "	" "
3	" "	"R"	"R"	"R"	"R"	" "	" "
4	" "	"R"	"R"	"Y"	"Y"	" "	" "
5	"Y"	"R"	"R"	"R"	"Y"	"Y"	"Y"



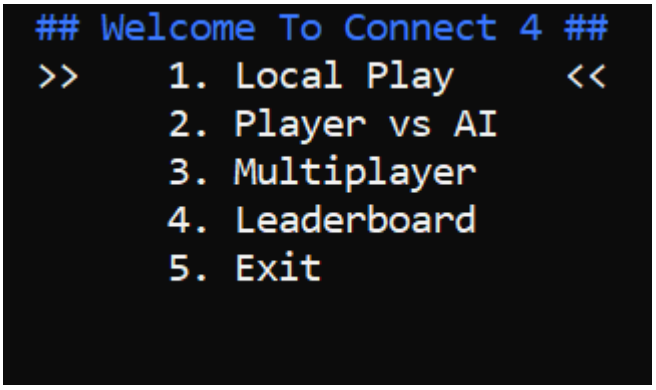
As you can see, the table is an abstraction of the 2D array and how each element can be represented by a “ “ for an empty space, a “Y” for a yellow token or a “R” for a red token. This is followed by how this board would appear within a game of connect 4, in this case it being the final state of the board after red achieved 4 in a row on the 3rd row. The contents of the array is refreshed after each game, so that it is once again populated, like it is initially, by just “ “ for each element.

2.3.5 User Interface (UI)

My project is an entirely console based project, meaning that all interaction the user has with the program is done through the console. While creating my project, I have tried to keep the user interface as simple and streamlined as possible. What this means is that the menu and other areas where the user

navigates for options, aren't cluttered and any desired options are very easy to find and access. Additionally, to minimise the number of written user inputs that are needed to be made, most areas of my code where a choice needs to be made are decided by the user highlighting the desired option and hitting enter, rather than typing out the option (See testing video at beginning). This makes the system easy to navigate and also reduces potential error within the program or from the user attempting to use it.

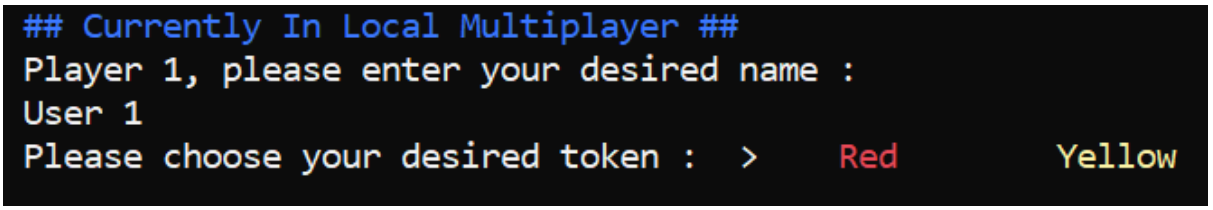
Main menu



To the left is the finalised main menu, which is the first and last screen the user will see while operating the program. The options are clearly labelled and accessible by using either the up and down arrows, or the W or S keys, and then pressing enter.

This main menu branches to 3 different areas of my code, as Player vs AI is unavailable and option 5 exits the program. These 3 areas are Local play, multiplayer or the leaderboard. The intention of these modes is very clear from their naming conventions. The order which these options take is fairly general, although I chose to make exiting the program the last option to help mitigate the ability to accidentally exit while trying to access a different feature.

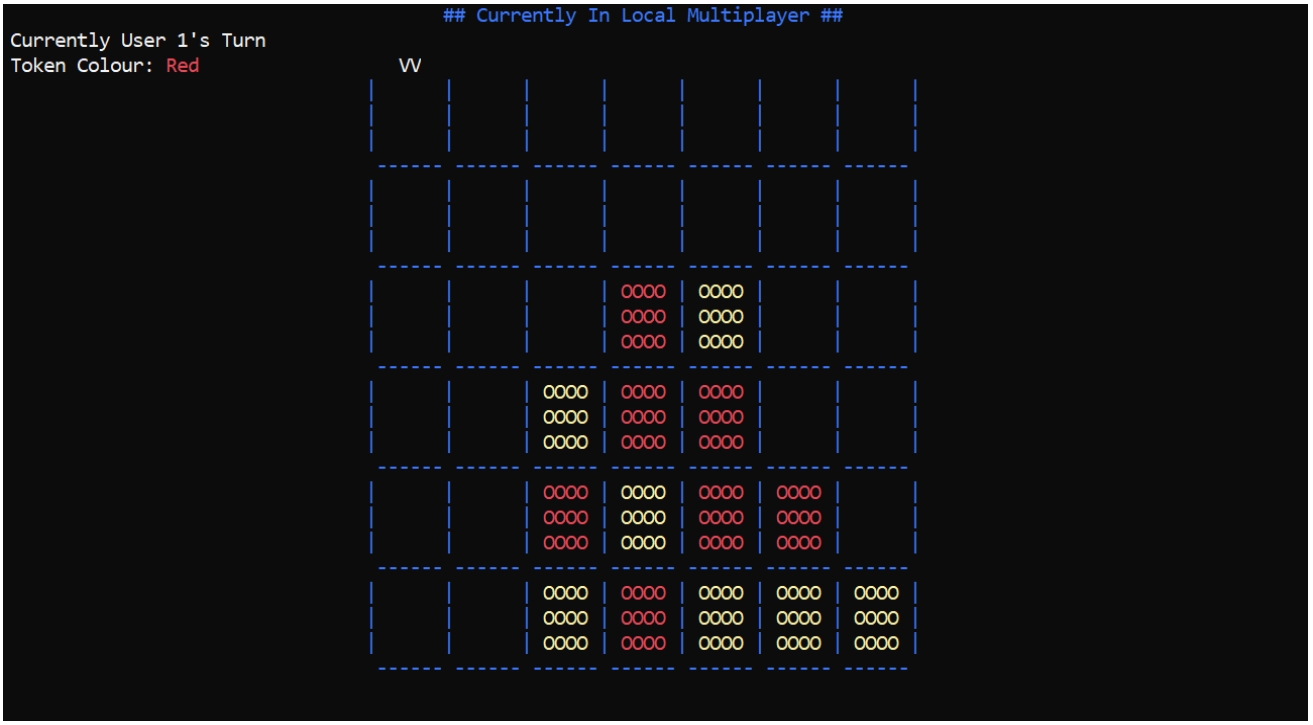
User Setup



This image demonstrates how a user may customise their name and token colour before a game starts. This part of user setup was taken from the local multiplayer mode although it is an identical process in the online multiplayer mode as well. The name had to obviously be created through a user defined text input, but the choice of token colour was once again reduced to just

navigation and selection. In this case you use either the left or right arrow keys or the A or D keys and press enter.

Board Display



Above we can see how the board and tokens are displayed to the user, and how the user interacts with the board. Once again through the use of either arrow keys or A/D, the user can select 1 of 7 columns to place their token in. The token is then placed at the lowest available point of that column. The use of an indicator above the currently selected column helps users keep track of where they are moving the indicator and which column the token will fall into if they press enter. Moreover, a note can be found in the top left of the console, notifying the player's that it is User 1's turn and also what their token colour is, in this case red. The same notice can be found in the right hand corner of the console on player 2's turn.

Rematch Screen

```
## Currently In Local Multiplayer ##

CONGRATULATIONS User 1 FOR WINNING!

Would you like a rematch?
Please choose Yes or No :          >    Yes        No
```

The rematch screen is another very simply displayed area of my program. The screen offers a congratulations to the winner and asks the user's if they would like to have a rematch. In local multiplayer, this is based off of just the choice shown here, however in multiplayer both players can make a decision through the program and both must agree for a rematch to occur.

Leaderboard

```
## Currently In Leaderboard ##

          1. User1  : 18
          2. User2  : 13
          3. TheMan  : 8
          4. NotTheMan  : 5
```

The leaderboard is where users can find the total wins of each player. Player's are obviously ranked numerically on the left hand side, with their names displayed in the centre and their wins displayed to the left. There is limited to no user interaction available on this screen, as there is nothing for the player to manually do, as the leaderboard automatically updates as the game is played. User's can exit this screen by simply pressing enter and it will return them to the main menu.

3 Testing

Testing Video URL:

https://tinyurl.com/2p8crbnh

Test No.	Description/ Purpose	Test Data	Expected Outcome	Pass/Fail	Evidence
1	Board scale (Objective 1.1.)	- Local Play “Alex” - Red “TestBot”	- Board successfully displayed	Pass	0:38 3:54
2	Board colour blue (Objective 1.2.)	- Local Play “Alex” - Red “TestBot”	- Board successfully displayed	Pass	0:38 3:54
3	Board spacing (Objective 1.3.)	- Local Play “Alex” - Red “TestBot”	- Board successfully displayed	Pass	0:38 3:54
4	Counter colour (Objective 2.1.)	- Local Play “Alex” - Red “TestBot”	- Counters successfully displayed	Pass	0:46
5	Counter size (Objective 2.2.)	- Local Play “Alex” - Red “TestBot”	- Counters successfully displayed	Pass	0:46
6	Host goes first (Objective 3.1.)	- Local Play “Alex” - Red “TestBot”	- Host takes first turn	Pass	0:40 3:54
7	Player’s choose token colours and	- Local Play “Alex” - Red “TestBot”	- Player 1 chooses 1 of 2 token colours, the	Pass	0:19 1:38 3:50

	name (Objective 3.2.)		other player receives the unchosen colour		
8	Counters may be placed (Objective 3.3.)	- Local Play “Alex” - Red “TestBot”	- Counters can be placed in 1 of 7 columns at the lowest available space	Pass	0:46 2:00 3:58
9	Player’s alternate turns (Objective 3.4.)	- Local Play “Alex” - Red “TestBot”	- Players take turns one at a time	Pass	0:44 2:00 3:58
10	Player wins when one achieves 4 in a row (Objective 3.5.)	- Local Play “Alex” - Red “TestBot”	- A player wins when they achieve 4 in a row	Pass	1:11 2:05 7:49 8:15
11	Client enters a host IP address (Objective 4.1.)	- Multiplayer “192.168.0.245” - Client	- Client enters the host IP address	Pass	4:48
12	Establish connection between client and server (Objective 4.2.)	- Multiplayer “192.168.0.245” - Client	- A connection is established between host and server	Pass	2:42 4:54
13	Allow users to choose preferences in multiplayer (Objective 4.3.)	- Multiplayer “192.168.0.245” - Host	- Each User can choose their preferences in multiplayer	Pass	3:35 5:15
14	Allow users to take turns one by one in	- Multiplayer “192.168.0.245” - Host	- Players can take their turns one at a time in	Pass	3:58 5:44

	multiplayer (Objective 4.4.)	“Alex” - Red	multiplayer from separate instances		
15	AI section of connect 4 (Objective 5.1.)	N/A	N/A	Fail	N/A
16	Apply Algorithms (Objective 5.2.)	N/A	N/A	Fail	N/A
17	User vs AI (Objective 5.3.)	N/A	N/A	Fail	N/A
18	Leaderboard accessible from menu (Objective 6.1.)	- Leaderboard	- User can access a leaderboard from the menu	Pass	0:14 2:10
19	Leaderboard updates upon opening (Objectives 6.2.)	- Leaderboard	- After playing any amount of games, the leaderboard is updated and sorted by wins in descending order	Pass	1:20 6:44 8:20
20	Add a user to the leaderboard if it is their first win (Objective 6.3.)	- Leaderboard	- New users are visible on the leaderboard with the correct number of wins displayed	Pass	2:10
21	Users can play with each other even while	- Local Play	- Users can play each other while on the same	Pass	0:40

	using the same pc (Objective 7.1.)		computer		
22	Rematch available after each game (Objective 7.2.)	- Local Play	- Rematch is available after each match	Pass	0:57 2:08 4:34 6:25
23	Whole system test	N/A	All modes (excluding A.I) fully accessible and functional	Pass	Entire Video

4 Evaluation

4.1 Overall Effectiveness of the System

The system meets most of the criteria set out in the original problem. The game itself plays in perfect accordance with the rules of Connect 4 The ability to play Connect 4 against an opponent, whether it is on the same computer/network or even against someone on a different network, which was the main goal of my project. This area of my program works flawlessly, with no errors, in particular within the networked portion, which is where I felt there was the most potential for issue. In spite of this, I failed to incorporate the intended A.I segment of my program. This was one of the desired features of my project and also the subject of a fair amount of my initial analysis. The lack of this feature is definitely the largest area of improvement for my project, as it was one of the main requested features. While I still believe the intent behind the problem has been solved for the most part, it is still the largest and most obvious refinement for my project and has a definite impact on the overall effectiveness. In summary, while the problem has been solved to a decent extent, it’s efficiency is limited by the lack of the A.I mode.

4.2 Evaluation of Objectives

Design a digital version of Connect 4 with multiplayer and AI capabilities, playable with one or two players.

1. **Design Connect 4 Board**
   See Testing, Test no's 1-3

1.1. Board of scale (7 x 6) (Columns x Rows).
Completed, the board has correct dimensions in all modes of play. For a console application the board looks decent, given more time could probably be made to look better by using a method other than creative character usage.

1.2. Board should be coloured Blue.
Same as with the board scale, edges of the board are coloured blue whenever displayed. This was a very simple objective just to make the program feel authentic with the original, physical game.

1.3. Board should have spaces to fit counters, which also correlates to the dimensions (7 columns of spaces, 6 rows of spaces).
Each column contains 6 separate spaces (6 rows), across the 7 columns. Each space is the exact same size and fits the tokens as designed.

2. **Design Playing Counters**
   See Testing, Test no's 4-5

2.1. Counters of two colours (Red and Yellow).
Token's of two colours, red or yellow, available to be chosen.

2.2. Counters should have dimensions that match board spaces, mustn't exceed or be shorter than.
Token's designed in such a way that they fit the spaces mentioned in objective 1.3. While they do fit the spaces, there are gaps between them and the board due to how characters were used in their design. This is a potential area of improvement when it comes to the look of the game.

3. **Connect 4 Rules**
   See Testing, Test no's 6-10

3.1. Host of the lobby (Player 1) Always goes first.
Completed, player 1 also has the choice of token colour.

3.2. Players may choose their token colour and name.

Done within the user setup section before each game starts. The customisations available being only name and token colour does feel lack-lustre. Possible area of improvement although there isnt alot to customise about a game like connect 4.

3.3. Counters can be placed into the board.

3.3.1 Counters can be placed down one of the 7 columns.

This has been completed perfectly, player's only have a choice of 1 of 7 columns to place a token into, without being able to make any other inputs during this time.

3.3.2 Counters must be placed at the lowest point of the column, which is either the bottom of the board or on top of any counters already within the column.

Similar to above, whenever player's select a column, the token is always placed at the lowest possible point, without fail.

3.4. Players will alternate turns, placing one counter each time.

Simple to complete but done just as described, player's exchange turns everytime one of them makes an input/move during their turn.

3.5. The game must end when either player achieves 4 of their counters in a row (This can be of any orientation).

This has been done by a self-made algorithm which analyses all possible ways to win to see if the board matches any of these. Could probably be vastly improved efficiency-wise as it is just checking all possible ways to win in all possible arrangements. However there appears to be no time delay between turns when this algorithm runs so it's improvement wouldn't have a notable impact on the user.

4. Networking Capabilities

See Testing, Test no's 11-14

4.1. Have the client enter an IP address of the host computer.

Client enters an IP address upon selecting the client option within the multiplayer section. Hosts also have to enter their own IP address which is a very redundant step which further complicates the user experience. A definite place for improvement would be to remove this step for the host entirely. It will always be necessary for the client in order to be able to connect to the server.

4.2. Establish connections between client and host.

Completed well, both parties involved are informed when a connection is established, which will occur shortly after the client enters the designated IP address for the host server.

4.3. Allow users to pick preferences.

This has been fulfilled, although the client must wait for the host to pick their preferences first before it is able to do anything, so a potential development to this process would be to allow both to do so at the same time.

4.4. Ensure users can control and take their turns one after the other.

This is the case, with each user taking their turns asynchronously, without either user being able to influence the inputs of the other.

5. **AI**

   As mentioned in the overall effectiveness of the system (4.1), I was unable to include an A.I mode to my program. This is by far the largest area of improvement for my project, as it is an entire concept which wasn't included within my program.

5.1. AI programmed with rules of connect 4.

5.2. Apply Algorithms (MinMax and Alpha-Beta Pruning).

5.3. Allows the user to play against AI.

6. **LeaderBoard**

   See Testing, Test no's 18-20

6.1. Create a leaderboard, accessible from the menu.

This is an available option within the main menu.

6.2. Have leaderboard update upon opening.

This has been completed perfectly using a merge sort algorithm which runs whenever the leaderboard is to be opened.

6.3. Add/Update a user depending on if they've already won or not.

This works fine with a user either being added to the leaderboard with 1 win if it's their first time, or their score is increased by 1 if they're already on the leaderboard.

7. **Local Play**

   See Testing, Test no's 21-22

7.1. User(s) able to play against each other from the same computer.

This works fully, quite a simple mode to include but it works flawlessly.

7.2. Rematch available after each game.

This is the case with all available game modes, not just local play, and works exactly as intended.

4.3 End User Feedback

Access to my program was made available to both of my original interviewees, who are also target end user's of my program to get feedback from both of them on the finalised program. Here's what they had to say:

End User 1:

"The game plays really nicely. For a console application, I think the design is great and it looks good and feels good when playing. The multiplayer works decently and wasn't as Intimidating as I was expecting to set up. There is, unfortunately, no A.I mode which is frustrating to be sure and would be a good way to keep people engaged with the game outside of just the multiplayer experience, especially seeing as it was one of the things I was expecting to see in the finished project. The leaderboard is cool, although i do personally find it redundant to have the multiplayer wins and local wins grouped together, i think it would be more innovative to have them as separate categories but that feels more minor compared to the complete lack of the A.I mode. This is definitely where my largest complaint with the program lies, but otherwise I think it does a good job with it's other features."

End User 2:

"Well I'll start with the positives. The ability to play against other player's works well, on my computer and also against (End User 1) *on a completely separate computer. The ability to rematch after each game, however many times you want is great, I don't want to have to navigate the menu every single time I play the same person again. Also, the leaderboard works decently, updates each time I win and is also sorted from most wins to least, so I can know when I'm better than everyone else.*

However, there are definitely things I would improve. I mean for starters, there was no A.I mode in the final program. This isn't a major issue, but it's a mode i was quite interested in trying as i haven't played against a bot in connect 4 before. I would've liked to have seen this feature and it is definitely a disappointment that it isn't available. Aside from what is

quite an obvious omission, I also think the leaderboard feels a bit … empty? I said previously that I like that it displays the wins for each player and that it is sorted, which I do like. But I feel like there are more things which can be incorporated into a leaderboard, like an actual skill rating, or a count for draws and losses as well, or something similar. Those are just my thoughts on it."

As was expected, the most glaring issue with my program is of course, the absence of the A.I mode. This is by far the main thing I could look to implement to elevate the standard of the program. I'm glad that the multiplayer was functioning how they expected and that the leaderboard was a success for the most part. I definitely agree that something more could be done to the leaderboard to make it more interesting. Potentially incorporating some of the suggestions that they made in their feedback would be a smart way of doing so.

4.4 System Improvements

First and foremost, the A.I mode. Incorporating this mode would be best done using the original methods I describe in the analysis section. Through the utilisation of a MinMax algorithm and an optimisation for it, Alpha-Beta pruning, I would create an A.I which could play against the user and provide a new and unique experience when compared with playing against another player. Even more so, it could be an A.I with varying difficulty levels, as mentioned by both interviewees in my first interview. Another useful improvement would be making the leaderboard more diverse and the suggestions by my end users of either skill ratings, or different categories or even both would be great additions to this feature. Smaller but still possible improvements would be a more efficient algorithm for detecting if a player has won the game, or making the user setup more interesting, through additional customisation options, maybe the colour of a user's name or even displaying their current win record.

5 Technical Solution

5.1 Contents page for code

5.2 Code

```
Imports System.IO
Module Module1
    Dim Board(5, 6) As String
    Sub Main()
      Console.cursorvisible = false
        Do
            For i = 0 To 5
                For j = 0 To 6
                    Board(i, j) = " "
                Next
            Next
            Select Case Menu()
                Case 1
                    LocalPlay()
```

```
                Case 2
                    AIPlay()
                Case 3
                    Multiplayer()
                Case 4
                    LeaderBoard()
                Case 5
                    Console.ForegroundColor = ConsoleColor.Blue
                    Writeline("## Thank You For Playing ##")
                    Threading.Thread.Sleep(2500)
                    End
            End Select
            Console.Clear()
        Loop
        Console.ReadKey()
    End Sub

    Function UserInfo(type As Integer, other As ConsoleColor) As User
        Console.CursorVisible = True
        Dim name As String
        Dim token As ConsoleColor
        Dim choice As ConsoleKey
        Dim position As Integer = 76
        Select Case type
            Case 1
                Writeline("Player 1, please enter your desired name : ")
                Console.SetCursorPosition(40, Console.CursorTop)
                name = Console.ReadLine()
                Console.CursorVisible = False
                Writeline("Please choose your desired token :")
                Console.ForegroundColor = ConsoleColor.Red
                Console.SetCursorPosition(position, Console.CursorTop -
1)
                Write("     Red     ")
                Console.ForegroundColor = ConsoleColor.Yellow
                Write("   Yellow   ")
                Console.ForegroundColor = ConsoleColor.White
                Do
                    Console.SetCursorPosition(position, 3)
                    Write(">")
                    choice = Console.ReadKey(True).Key
                    If choice = ConsoleKey.A And position <> 76 Or
choice = ConsoleKey.LeftArrow And position <> 76 Then
                        Console.SetCursorPosition(position, 3)
                        Write(" ")
                        position = 76
                    ElseIf choice = ConsoleKey.D And position <> 87 Or
```

```
choice = ConsoleKey.RightArrow And position <> 87 Then
                        Console.SetCursorPosition(position, 3)
                        Write(" ")
                        position = 87
                    End If
                Loop Until choice = ConsoleKey.Enter
                If position = 76 Then
                    token = ConsoleColor.Red
                ElseIf position = 87 Then
                    token = ConsoleColor.Yellow
                End If
            Case 2
                Writeline("Player 2, please enter your desired name : ")
                Console.SetCursorPosition(40, Console.CursorTop)
                name = Console.ReadLine()
                Console.CursorVisible = False
                Writeline("You will be attributed the other token : ")
                Console.SetCursorPosition(82, Console.CursorTop - 1)
                If other = 12 Then
                    Console.ForegroundColor = ConsoleColor.Yellow
                    token = ConsoleColor.Yellow
                    Write("Yellow   ")
                ElseIf other = 14 Then
                    Console.ForegroundColor = ConsoleColor.Red
                    token = ConsoleColor.Red
                    Write("Red     ")
                End If
                Threading.Thread.Sleep(2000)
        End Select
        ClearLines()
        Return New User(name, token)
    End Function

    Sub ClearLines()
        Dim position As Integer = Console.WindowHeight
        Do Until position = 0
            Console.SetCursorPosition(0, position)
            Write(New String(" ", Console.BufferWidth))
            Console.SetCursorPosition(0, position)
            position -= 1
        Loop
        Console.SetWindowPosition(0, 0)
    End Sub

    Function GameWon(player As User) As Boolean
        Dim position As Integer = 76
        Dim choice As ConsoleKey
```

```
            AddToLeaderBoard(player)
            ClearLines()
            Writeline("")
            Writeline("CONGRATULATIONS " & player.name & " FOR WINNING! ")
            Console.ForegroundColor = ConsoleColor.White
            Writeline("")
            Writeline("Would you like a rematch?")
            Console.CursorVisible = False
            Writeline("Please choose Yes or No :")
            Console.SetCursorPosition(position, Console.CursorTop - 1)
            Write("     Yes     ")
            Write("   No   ")
            Console.ForegroundColor = ConsoleColor.White
            Do
                Console.SetCursorPosition(position, 5)
                Write(">")
                choice = Console.ReadKey(True).Key
                If choice = ConsoleKey.A And position <> 76 Or choice =
ConsoleKey.LeftArrow And position <> 76 Then
                    Console.SetCursorPosition(position, 5)
                    Write(" ")
                    position = 76
                ElseIf choice = ConsoleKey.D And position <> 87 Or choice =
ConsoleKey.RightArrow And position <> 87 Then
                    Console.SetCursorPosition(position, 5)
                    Write(" ")
                    position = 87
                End If
            Loop Until choice = ConsoleKey.Enter
            If position = 76 Then
                Return True
            ElseIf position = 87 Then
                Return False
            End If
            Return False
        End Function

        Sub TraverseBoard(Current As User)
            Console.CursorSize = 100
            Console.SetCursorPosition(36, 2)
            Dim press As ConsoleKey
            Dim count As Integer = 0
            Console.SetCursorPosition(Console.CursorLeft, Console.CursorTop)
            Write("VV")
            Do
                press = Console.ReadKey(True).Key
                Console.SetCursorPosition(Console.CursorLeft - 2,
```

```
Console.CursorTop)
            Write("  ")
            Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
            If press = ConsoleKey.A Or press = ConsoleKey.LeftArrow Then
                If count <> 0 Then
                    Console.SetCursorPosition(Console.CursorLeft - 7,
Console.CursorTop)
                    count -= 1
                End If
            ElseIf press = ConsoleKey.D Or press = ConsoleKey.RightArrow
Then
                If count <> 6 Then
                    Console.SetCursorPosition(Console.CursorLeft + 7,
Console.CursorTop)
                    count += 1
                End If
            End If
            Console.SetCursorPosition(Console.CursorLeft,
Console.CursorTop)
            Write("VV")
        Loop Until press = ConsoleKey.Enter
        Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
        Write("  ")
        If Current.token = ConsoleColor.Red Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "R"
                    Exit For
                End If
            Next
        ElseIf Current.token = ConsoleColor.Yellow Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "Y"
                    Exit For
                End If
            Next
        End If
    End Sub

    Sub DisplayBoard()
        Writeline("")
        Console.ForegroundColor = ConsoleColor.Blue
        For i = 5 To 0 Step -1
            Writeline("")
```

```
Console.SetCursorPosition(33, Console.CursorTop)
For j = 0 To 6
    If j = 0 Then
        Write("|")
    End If
    If Board(i, j) = "R" Then
        Console.ForegroundColor = ConsoleColor.Red
        Write(" OOOO ")
    ElseIf Board(i, j) = "Y" Then
        Console.ForegroundColor = ConsoleColor.Yellow
        Write(" OOOO ")
    Else
        Write("      ")
    End If
    Console.ForegroundColor = ConsoleColor.Blue
    Write("|")
Next
Writeline("")
Console.SetCursorPosition(33, Console.CursorTop)
For j = 0 To 6
    If j = 0 Then
        Write("|")
    End If
    If Board(i, j) = "R" Then
        Console.ForegroundColor = ConsoleColor.Red
        Write(" OOOO ")
    ElseIf Board(i, j) = "Y" Then
        Console.ForegroundColor = ConsoleColor.Yellow
        Write(" OOOO ")
    Else
        Write("      ")
    End If
    Console.ForegroundColor = ConsoleColor.Blue
    Write("|")
Next
Writeline("")
Console.SetCursorPosition(33, Console.CursorTop)
For j = 0 To 6
    If j = 0 Then
        Write("|")
    End If
    If Board(i, j) = "R" Then
        Console.ForegroundColor = ConsoleColor.Red
        Write(" OOOO ")
    ElseIf Board(i, j) = "Y" Then
        Console.ForegroundColor = ConsoleColor.Yellow
        Write(" OOOO ")
```

```
            Else
                Write("      ")
            End If
            Console.ForegroundColor = ConsoleColor.Blue
            Write("|")
        Next
        Writeline("")
        Console.SetCursorPosition(34, Console.CursorTop)
        Write("------ ------ ------ ------ ------ ------ ------")
    Next
    Console.ForegroundColor = ConsoleColor.White
End Sub

Function Merge(first() As Integer, Second() As Integer)
    Dim final(first.Length - 1 + Second.Length) As Integer
    Dim temp(final.Length - 1)
    Dim counter As Integer
    Dim current As Integer
    Dim firstholder() As Integer = first
    Dim secondholder() As Integer = Second
    Do
        For i = 0 To first.Length - 1
            For j = 0 To Second.Length - 1
                If first(i) < Second(j) Then
                    counter += 1
                ElseIf first(i) = Second(j) Then
                    counter += 1
                End If
            Next
            If counter = Second.Length Then
                counter = 0
                final(current) = first(i)
                current += 1
                If first.Length = 1 And Second.Length = 1 Then
                    final(current) = Second(0)
                    current += 1
                    Exit Do
                ElseIf first.Length = 1 Then
                    For o = 0 To Second.Length - 1
                        final(current) = Second(o)
                        current += 1
                    Next
                    Exit Do
                Else
                    For k = i + 1 To first.Length - 1
                        firstholder(k - 1) = first(k)
                        ReDim first(firstholder.Length - 2)
```

```
                    For l = 1 To firstholder.Length - 1
                        first(l - 1) = firstholder(l)
                    Next
                    ReDim firstholder(first.Length - 1)
                    For l = 0 To firstholder.Length - 1
                        firstholder(l) = first(l)
                    Next
                    Exit For
                Next
                Exit For
            End If
        End If
        counter = 0
    Next
    For i = 0 To Second.Length - 1
        For j = 0 To first.Length - 1
            If Second(i) < first(j) Then
                counter += 1
            ElseIf Second(i) = first(j) Then
                counter += 1
            End If
        Next
        If counter = first.Length Then
            counter = 0
            final(current) = Second(i)
            current += 1
            If Second.Length = 1 And first.Length = 1 Then
                final(current) = first(0)
                current += 1
                Exit For
            ElseIf Second.Length = 1 Then
                For o = 0 To first.Length - 1
                    final(current) = first(o)
                    current += 1
                Next
                Exit Do
            Else
                For k = i + 1 To Second.Length - 1
                    secondholder(k - 1) = Second(k)
                    ReDim Second(secondholder.Length - 2)
                    For l = 1 To secondholder.Length - 1
                        Second(l - 1) = secondholder(l)
                    Next
                    ReDim secondholder(Second.Length - 1)
                    For l = 0 To secondholder.Length - 1
                        secondholder(l) = Second(l)
                    Next
```

```
                            Exit For
                        Next
                        Exit For
                    End If
                End If
                counter = 0
            Next
        Loop Until current = final.Length
        Return final
    End Function

    Function MergeSort(numbers() As Integer, pointer As Integer)
        Dim first(pointer - 1) As Integer
        Dim second(numbers.Length - 1 - pointer) As Integer
        Dim firstpointer As Integer = (first.Length) / 2
        Dim secondpointer As Integer = (second.Length) / 2
        For i = 0 To first.Length - 1
            first(i) = numbers(i)
        Next
        For i = first.Length To numbers.Length - 1
            second(i - first.Length) = numbers(i)
        Next
        If first.Length > 1 Then
            first = MergeSort(first, firstpointer)
        End If
        If second.Length > 1 Then
            second = MergeSort(second, secondpointer)
        End If
        Dim solution() As Integer = Merge(first, second)
        Return solution
    End Function

    Sub UpdateLeaderBoard()
        Dim name As String = "Leaderboard.txt"
        Dim contents() As String = File.ReadAllLines(name)
        Dim valuesstring(contents.Length - 1) As String
        Dim position As Integer
        Dim values(contents.Length - 1) As Integer
        Dim midpoint As Integer = System.Math.Ceiling((contents.Length -
1) / 2)
        Dim order As New List(Of Integer)
        For i = 0 To contents.Length - 1
            position = InStrRev(contents(i), " ")
            valuesstring(i) =
StrReverse(contents(i)).Remove(contents(i).Length - position)
            valuesstring(i) = StrReverse(valuesstring(i))
            values(i) = valuesstring(i)
```

```
        Next
        Dim sorted() As Integer = MergeSort(values, midpoint)
        Dim tempsorted(sorted.Length - 1) As Integer
        Dim tempcontents(contents.Length - 1) As String
        For i = 0 To sorted.Length - 1
            tempsorted(i) = sorted(i)
        Next
        For i = 0 To sorted.Length - 1
            sorted(i) = tempsorted(sorted.Length - 1 - i)
            order.Add(sorted(i))
        Next
        For i = 0 To order.Count - 1
            For j = 0 To values.Length - 1
                If order(i) = values(j) Then
                    tempcontents(i) = contents(j)
                End If
            Next
        Next
        Using writer As StreamWriter = New StreamWriter(name)
            For i = 0 To tempcontents.Length - 1
                writer.WriteLine(tempcontents(i))
            Next
        End Using
    End Sub

    Sub AddToLeaderBoard(player As User)
        Dim name As String = "Leaderboard.txt"
        Dim contents() As String = File.ReadAllLines(name)
        Dim wins As String
        Dim winscount As Integer
        Dim newwinner As String = ""
        Dim position As Integer
        Dim temp As Integer
        Dim playername As String
        Dim found As Boolean
        If contents.Length <> 0 Then
            For i = 0 To contents.Length - 1
                position = InStrRev(contents(i), " ")
                wins = Mid(contents(i), position + 1)
                winscount = wins
                temp = contents(i).Length - position + 1
                playername = contents(i).Remove(position - 1, temp)
                If playername = player.name Then
                    winscount += 1
                    contents(i) = playername & " " & winscount
                    found = True
                End If
```

```
            Next
        Else
            newwinner = player.name & " " & 1
        End If
        If found = False Then
            newwinner = player.name & " " & 1
        End If
        Using writer As StreamWriter = New StreamWriter(name)
            For i = 0 To contents.Length - 1
                writer.WriteLine(contents(i))
            Next
            If newwinner <> "" Then
                writer.WriteLine(newwinner)
            End If
        End Using
    End Sub

    Sub LeaderBoard()
        UpdateLeaderBoard()
        Dim name As String = "Leaderboard.txt"
        Dim lineCount As Integer = File.ReadAllLines(name).Length
        Dim line(lineCount) As String
        Dim user(lineCount) As String
        Dim wins(lineCount) As Integer
        Dim count As Integer
        Dim position As Integer
        Dim temp As String
        Console.CursorVisible = False
        Console.ForegroundColor = ConsoleColor.Blue
        Writeline("## Currently In Leaderboard ##")
        Console.ForegroundColor = ConsoleColor.Cyan
        Writeline("")
        Using reader As StreamReader = New StreamReader(name)
            Do Until reader.EndOfStream
                line(count) = reader.ReadLine()
                count += 1
            Loop
        End Using
        count = 0
        For i = 0 To lineCount - 1
            temp = StrReverse(line(i))
            position = InStr(temp, " ")
            user(i) = StrReverse(Mid(temp, position))
            position = InStrRev(line(i), " ")
            wins(i) = Mid(line(i), position)
            Console.SetCursorPosition(50, Console.CursorTop)
            Write(count + 1 & ". " & user(i) & " : " & wins(i))
```

```
            count += 1
            Writeline("")
        Next
        Console.ReadKey()
    End Sub

    Function OnlineSetup()
        Dim position As Integer = 45
        Dim choice As ConsoleKey
        Writeline("Please may whoever is hosting choose first!")
        Writeline("Please choose either Host or Client: ")
        Console.WriteLine("")
        Console.CursorVisible = False
        Console.SetCursorPosition(45, Console.CursorTop)
        Write("   Host    ")
        Console.SetCursorPosition(55, Console.CursorTop)
        Write("   Client  ")
        Do
            Console.SetCursorPosition(position, Console.CursorTop)
            Write(">")
            choice = Console.ReadKey(True).Key
            If choice = ConsoleKey.A And position <> 45 Or choice =
ConsoleKey.LeftArrow And position <> 45 Then
                Console.SetCursorPosition(position, Console.CursorTop)
                Write(" ")
                position = 45
            ElseIf choice = ConsoleKey.D And position <> 55 Or choice =
ConsoleKey.RightArrow And position <> 55 Then
                Console.SetCursorPosition(position, Console.CursorTop)
                Write(" ")
                position = 55
            End If
        Loop Until choice = ConsoleKey.Enter
        ClearLines()
        Console.CursorVisible = True
        Select Case position
            Case 45
                Return 1
            Case Else
                Return 2
        End Select
    End Function

    Sub OnlineTraverseBoard1(Current As User, choice As Integer, System
As OnlineServer)
        Console.CursorSize = 100
        Console.SetCursorPosition(36, 2)
```

```
        Dim press As ConsoleKey
        Dim count As Integer = 0
        Console.SetCursorPosition(Console.CursorLeft, Console.CursorTop)
        Write("VV")
        Do
            If choice = 0 Then
                press = Console.ReadKey(True).Key
                System.Send(press)
                System.Check = False
            ElseIf choice = 1 Then
                Do
                    Threading.Thread.Sleep(100)
                Loop Until System.Check = True
                press = System.Data
                System.Check = False
            End If
            Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
            Write("  ")
            Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
            If press = ConsoleKey.A Or press = ConsoleKey.LeftArrow Then
                If count <> 0 Then
                    Console.SetCursorPosition(Console.CursorLeft - 7,
Console.CursorTop)
                    count -= 1
                End If
            ElseIf press = ConsoleKey.D Or press = ConsoleKey.RightArrow
Then
                If count <> 6 Then
                    Console.SetCursorPosition(Console.CursorLeft + 7,
Console.CursorTop)
                    count += 1
                End If
            End If
            Console.SetCursorPosition(Console.CursorLeft,
Console.CursorTop)
            Write("VV")
        Loop Until press = ConsoleKey.Enter
        Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
        Write("  ")
        If Current.token = ConsoleColor.Red Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "R"
                    Exit For
```

```
                End If
            Next
        ElseIf Current.token = ConsoleColor.Yellow Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "Y"
                    Exit For
                End If
            Next
        End If
        System.Check = False
    End Sub

    Sub OnlineTraverseBoard2(Current As User, choice As Integer, System
As OnlineClient)
        Console.CursorSize = 100
        Console.SetCursorPosition(36, 2)
        Dim press As ConsoleKey
        Dim count As Integer = 0
        Console.SetCursorPosition(Console.CursorLeft, Console.CursorTop)
        Write("VV")
        Do
            If choice = 0 Then
                press = Console.ReadKey(True).Key
                System.Send(press)
                System.Check = False
            ElseIf choice = 1 Then
                Do
                    Threading.Thread.Sleep(100)
                Loop Until System.Check = True
                press = System.Data
                System.Check = False
            End If
            Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
            Write("  ")
            Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
            If press = ConsoleKey.A Or press = ConsoleKey.LeftArrow Then
                If count <> 0 Then
                    Console.SetCursorPosition(Console.CursorLeft - 7,
Console.CursorTop)
                    count -= 1
                End If
            ElseIf press = ConsoleKey.D Or press = ConsoleKey.RightArrow
Then
                If count <> 6 Then
```

```
                    Console.SetCursorPosition(Console.CursorLeft + 7,
Console.CursorTop)
                    count += 1
                End If
            End If
            Console.SetCursorPosition(Console.CursorLeft,
Console.CursorTop)
            Write("VV")
        Loop Until press = ConsoleKey.Enter
        Console.SetCursorPosition(Console.CursorLeft - 2,
Console.CursorTop)
        Write("  ")
        If Current.token = ConsoleColor.Red Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "R"
                    Exit For
                End If
            Next
        ElseIf Current.token = ConsoleColor.Yellow Then
            For i = 0 To 5
                If Board(i, count) = " " Then
                    Board(i, count) = "Y"
                    Exit For
                End If
            Next
        End If
        System.Check = False
    End Sub

    Sub Multiplayer()
        Dim ServerControl As OnlineServer
        Dim ClientControl As OnlineClient
        Dim IP As String
        Dim Player1 As User
        Dim Player2 As User
        Dim MultiUser(1) As String
        Dim winner As User
        Dim rematch As Boolean
        Console.ForegroundColor = ConsoleColor.Blue
        Writeline("## Currently In Online Multiplayer ##")
        Console.ForegroundColor = ConsoleColor.White
        Writeline("Please enter the IP address of the host system: ")
        Console.SetCursorPosition(40, Console.CursorTop)
        IP = Console.ReadLine
        ClearLines()
        Select Case OnlineSetup()
```

```
            Case 1
                ServerControl = New OnlineServer(IP)
                Writeline("----")
                Writeline("Server Successfully Started!")
                Writeline("----")
                AddHandler ServerControl.Receiver, AddressOf
ServerReceived
                Threading.Thread.Sleep(2000)
                ClearLines()
                Player1 = UserInfo(1, 0)
                Writeline("Waiting for client to connect...")
                Do
                    Threading.Thread.Sleep(200)
                Loop Until ServerControl.Connected = True
                ClearLines()
                ServerControl.Send(Player1.name & " " & Player1.token &
vbCrLf)
                Console.CursorVisible = False
                Writeline("Please wait for the Client to choose their
preferences...")
                Do
                    Threading.Thread.Sleep(100)
                Loop Until ServerControl.Check = True
                MultiUser(0) = ServerControl.Data
                If Player1.token = 12 Then
                    Player2 = New User(MultiUser(0), 14)
                ElseIf Player1.token = 14 Then
                    Player2 = New User(MultiUser(0), 12)
                End If
                ServerControl.Check = False
                Do
                    ClearLines()
                    If rematch = True Then
                        ServerControl.Send("Yes")
                        Writeline("Waiting for opponents rematch
response...")
                        Do
                            Threading.Thread.Sleep(100)
                        Loop Until ServerControl.Check = True
                        If ServerControl.Data = "No" Then
                            ServerControl.Check = False
                            Exit Do
                        End If
                    End If
                    ServerControl.Check = False
                    ClearLines()
                    For i = 0 To 5
```

```
                    For j = 0 To 6
                        Board(i, j) = " "
                    Next
                Next
                DisplayBoard()
                Do
                    Console.SetCursorPosition(1, 1)
                    Player1.Turn()
                    OnlineTraverseBoard1(Player1, 0, ServerControl)
                    ClearLines()
                    If CheckWon(Player1.token) = True Then
                        winner = Player1
                        Exit Do
                    End If
                    DisplayBoard()
                    Console.SetCursorPosition(94, 1)
                    Player2.Turn()
                    OnlineTraverseBoard1(Player2, 1, ServerControl)
                    ClearLines()
                    If CheckWon(Player2.token) Then
                        winner = Player2
                        Exit Do
                    End If
                    DisplayBoard()
                Loop
                DisplayBoard()
                Threading.Thread.Sleep(2000)
                rematch = GameWon(winner)
            Loop Until rematch = False
        Case 2
            ClientControl = New OnlineClient(IP)
            If ClientControl.Client.Connected Then
                Writeline("Connected")
            End If
            AddHandler ClientControl.Receiver, AddressOf
ClientReceived
            Threading.Thread.Sleep(2000)
            ClearLines()
            Console.CursorVisible = False
            Writeline("Please wait for the Host to choose their
preferences...")
            Do
                Threading.Thread.Sleep(100)
            Loop Until ClientControl.Check = True
            MultiUser = ClientControl.Data.Split(" ")
            Player1 = New User(MultiUser(0), MultiUser(1))
            ClientControl.Check = False
```

```
                ClearLines()
                Player2 = UserInfo(2, MultiUser(1))
                ClientControl.Send(Player2.name & vbCrLf)
                ClientControl.Check = False
                Do
                    ClearLines()
                    If rematch = True Then
                        ClientControl.Send("Yes")
                        Writeline("Waiting for opponents rematch
response...")
                        Do
                            Threading.Thread.Sleep(100)
                        Loop Until ClientControl.Check = True
                        If ClientControl.Data = "No" Then
                            ClientControl.Check = False
                            Exit Do
                        End If
                    End If
                    ClientControl.Check = False
                    ClearLines()
                    For i = 0 To 5
                        For j = 0 To 6
                            Board(i, j) = " "
                        Next
                    Next
                    DisplayBoard()
                    Do
                        Console.SetCursorPosition(1, 1)
                        Player1.Turn()
                        OnlineTraverseBoard2(Player1, 1, ClientControl)
                        ClearLines()
                        If CheckWon(Player1.token) = True Then
                            winner = Player1
                            Exit Do
                        End If
                        DisplayBoard()
                        Console.SetCursorPosition(94, 1)
                        Player2.Turn()
                        OnlineTraverseBoard2(Player2, 0, ClientControl)
                        ClearLines()
                        If CheckWon(Player2.token) Then
                            winner = Player2
                            Exit Do
                        End If
                        DisplayBoard()
                    Loop
                    DisplayBoard()
```

```
                Threading.Thread.Sleep(2000)
                rematch = GameWon(winner)
            Loop Until rematch = False
        End Select
        DisplayBoard()
    End Sub

    Sub ServerReceived(System As OnlineServer, Data As String)
        System.Check = True
        System.Data = Data
    End Sub

    Sub ClientReceived(System As OnlineClient, Data As String)
        System.Check = True
        System.Data = Data
    End Sub
    Sub AIPlay()

    End Sub

    Function CheckWon(token As ConsoleColor) As Boolean
        Dim letter As String
        If token = 12 Then
            letter = "R"
        Else
            letter = "Y"
        End If
        Dim count As Integer
        For k = 0 To 5
            For i = 0 To 3
                For j = 0 To 3
                    If Board(k, j + i) = letter Then
                        count += 1
                    End If
                Next
                If count = 4 Then
                    Return True
                End If
                count = 0
            Next
        Next
        For k = 0 To 6
            For i = 0 To 2
                For j = 0 To 3
                    If Board(j + i, k) = letter Then
                        count += 1
                    End If
```

```
                Next
                If count = 4 Then
                    Return True
                End If
                count = 0
            Next
        Next
        For k = 0 To 2
            For i = 0 To 6
                If Board(k + count, i) = letter Then
                    count += 1
                End If
                If count = 4 Then
                    Return True
                End If
            Next
            count = 0
        Next
        For k = 0 To 2
            For i = 6 To 0 Step -1
                If Board(k + count, i) = letter Then
                    count += 1
                End If
                If count = 4 Then
                    Return True
                End If
            Next
            count = 0
        Next
        Return False
    End Function

    Sub LocalPlay()
        Dim winner As User
        Console.ForegroundColor = ConsoleColor.Blue
        Writeline("## Currently In Local Multiplayer ##")
        Console.ForegroundColor = ConsoleColor.White
        Dim Player1 As User = UserInfo(1, 0)
        Dim Player2 As User = UserInfo(2, Player1.token)
        Do
            ClearLines()
            For i = 0 To 5
                For j = 0 To 6
                    Board(i, j) = " "
                Next
            Next
            DisplayBoard()
```

```
            Do
                Console.SetCursorPosition(1, 1)
                Player1.Turn()
                TraverseBoard(Player1)
                ClearLines()
                If CheckWon(Player1.token) = True Then
                    winner = Player1
                    Exit Do
                End If
                DisplayBoard()
                Console.SetCursorPosition(94, 1)
                Player2.Turn()
                TraverseBoard(Player2)
                ClearLines()
                If CheckWon(Player2.token) Then
                    winner = Player2
                    Exit Do
                End If
                DisplayBoard()
            Loop
            DisplayBoard()
            Threading.Thread.Sleep(2000)
        Loop Until GameWon(winner) = False
    End Sub

    Function Menu() As Integer
        Console.CursorVisible = False
        Console.ForegroundColor = ConsoleColor.Blue
        Writeline("## Welcome To Connect 4 ##")
        Console.ForegroundColor = ConsoleColor.White
        Writeline("      1. Local Play      ")
        Writeline("      2. Player vs AI      ")
        Writeline("      3. Multiplayer      ")
        Writeline("      4. Leaderboard      ")
        Writeline("      5. Exit       ")
        Dim position As Integer = 1
        Console.SetCursorPosition(40, position)
        Write(">>")
        Console.SetCursorPosition(64, position)
        Write("<<")
        Dim choice As ConsoleKey
        Do
            Console.SetCursorPosition(40, position)
            choice = Console.ReadKey(True).Key
            If choice = ConsoleKey.S And position < 5 Or choice =
ConsoleKey.DownArrow And position < 5 Then
                position += 1
```

```
                Write("  ")
                Console.SetCursorPosition(64, position - 1)
                Write("  ")
                Console.SetCursorPosition(40, position)
                Write(">>")
                Console.SetCursorPosition(64, position)
                Write("<<")
            ElseIf choice = ConsoleKey.W And position > 1 Or choice =
ConsoleKey.UpArrow And position > 1 Then
                position -= 1
                Write("  ")
                Console.SetCursorPosition(64, position + 1)
                Write("  ")
                Console.SetCursorPosition(40, position)
                Write(">>")
                Console.SetCursorPosition(64, position)
                Write("<<")
            End If
        Loop Until choice = ConsoleKey.Enter
        Console.CursorVisible = True
        Console.Clear()
        Return position
    End Function

    Sub Write(text As String)
        Console.SetCursorPosition(Console.CursorLeft, Console.CursorTop)
        Console.Write(text)
    End Sub

    Sub Writeline(text As String)
        Console.SetCursorPosition(40, Console.CursorTop)
        Console.WriteLine(text)
    End Sub
End Module



Public Class User
    Public name As String
    Public token As ConsoleColor
    Private position As Integer

    Sub New(input1 As String, input2 As ConsoleColor)
        name = input1
        token = input2
    End Sub
```

```
    Sub Turn()
        position = Console.CursorLeft
        Write("Currently " & name & "'s Turn")
        Writeline("")
        Console.SetCursorPosition(position, Console.CursorTop)
        Write("Token Colour: ")
        Console.ForegroundColor = token
        If token = 12 Then
            Write("Red")
        Else
            Write("Yellow")
        End If
        Console.ForegroundColor = ConsoleColor.White
    End Sub
End Class



Imports System.IO
Imports System.Net
Imports System.Net.Sockets
Imports System.Threading
Public Class OnlineServer
    Private ServerIP As IPAddress
    Private ServerPort As Integer = 1234
    Private Server As TcpListener
    Private ServerWriter As StreamWriter
    Private ServerReader As StreamReader
    Private Client As TcpClient
    Private ReadThread As Thread
    Public Check As Boolean
    Public Data As String
    Public Connected As Boolean

    Public Event Receiver(System As OnlineServer, Data As String)

    Public Sub New(IP As String)
        ServerIP = IPAddress.Parse(IP)
        Server = New TcpListener(ServerIP, ServerPort)
        Server.Start()

        ReadThread = New Thread(New ThreadStart(AddressOf Listening))
        ReadThread.Start()
    End Sub

    Public Sub Listening()
        Do
```

```
            If Server.Pending = True Then
                Client = Server.AcceptTcpClient
                ServerReader = New StreamReader(Client.GetStream)
                ServerWriter = New StreamWriter(Client.GetStream)
                Connected = True
            End If
            Try
                RaiseEvent Receiver(Me, ServerReader.ReadLine)
            Catch ex As Exception

            End Try
            Thread.Sleep(100)
        Loop
    End Sub

    Public Sub Send(Data As String)
        ServerWriter.Write(Data & vbCrLf)
        ServerWriter.Flush()
    End Sub
End Class



Imports System.IO
Imports System.Net
Imports System.Net.Sockets
Imports System.Threading
Public Class OnlineClient
    Private ClientWriter As StreamWriter
    Private ClientReader As StreamReader
    Public Client As TcpClient
    Private ServerPort As Integer = 1234
    Private ReadThread As Thread
    Public Check As Boolean
    Public Data As String

    Public Event Receiver(System As OnlineClient, Data As String)

    Public Sub New(IP As String)
        Client = New TcpClient(IP, ServerPort)
        ClientReader = New StreamReader(Client.GetStream)
        ClientWriter = New StreamWriter(Client.GetStream)
        ReadThread = New Thread(New ThreadStart(AddressOf Reading))
        ReadThread.Start()
    End Sub

    Public Sub Reading()
```

```
        Do
            Try
                RaiseEvent Receiver(Me, ClientReader.ReadLine)
            Catch ex As Exception

            End Try
            Thread.Sleep(100)
        Loop
    End Sub

    Public Sub Send(Data As String)
        ClientWriter.Write(Data & vbCrLf)
        ClientWriter.Flush()
    End Sub
End Class
```