NEA Project

Hex

Centre: Godalming college

Centre Number:64395

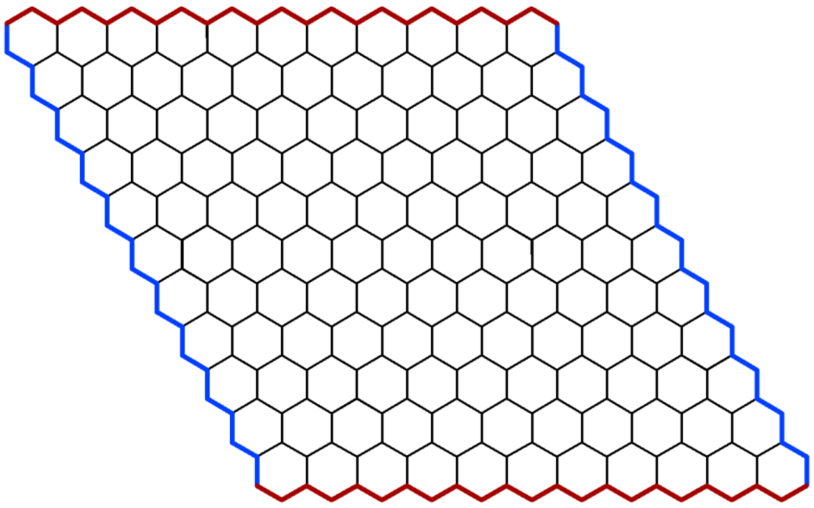
Cameron Zack

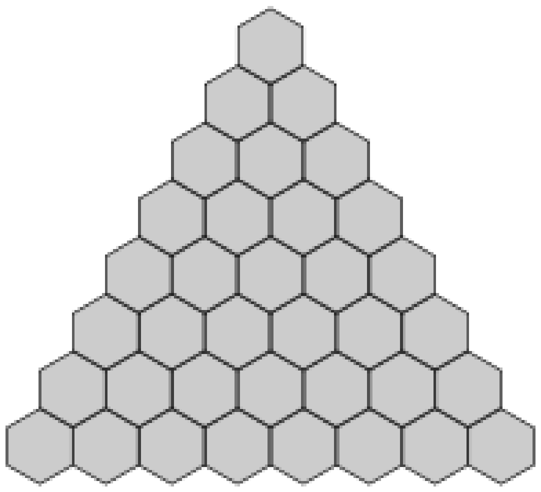
Contents

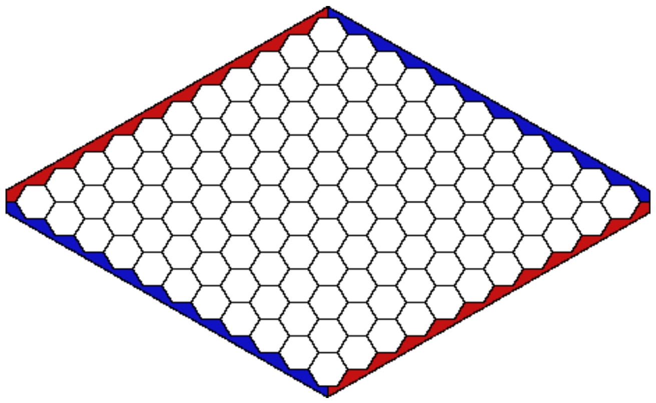
NEA research

Hex is a strategy board game that is played by two players. It can be played on a piece of paper using a pen or on a board using hexagonal tiles. It is typically played on a 11 by 11 rhombus made up of hexagonal shapes, the board can theoretically be of any size or a multitude of shapes. The other typical sizes of board are 13 by 13 and 19 by 19. However one of the games inventors thinks that a 14 by 14 grid would be the optimal size.

There are multiple variants on the hex game one where the hex tiles are replaced with rectangles this means that there can be no diagonal directions gone so must be vertical and horizontal. Another variation on the game is Misère hex where you want your opponent to make a chain. Another variation is Y which is played on a triangular grid of hexagons and the object is to connect all three sides of the triangle Y.


A typical 11 by 11 hex grid


A Y game board


A rectangular hex board

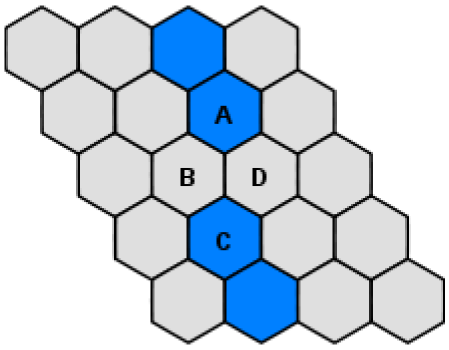
Game Play

Each player has an allocated colour, conventionally Red and Blue or White and Black but can be any two different colours. Players take turns placing a stone of their colour on a single cell within the overall playing board. Once placed, stones are not moved, captured or removed from the board. The goal for each player is to form a connected path of their own stones linking the opposing sides of the board marked by their colours, before their opponent connects his or her sides in a similar fashion. The first player to complete his or her connection wins the game. The four corner hexagons each belong to both adjacent sides. The game does not need to have all the spaces filled for a game to be won.

Strategies

There are a wide range of strategies to Hex as it has a complex type of connectivity. Play consists of creating small patterns which have a simpler type of connectivity called safely connected. Joining these safely connected patterns creates a path and eventually a player will end up creating a path of tiles of their colour to his opponent's side.

Different Strategies



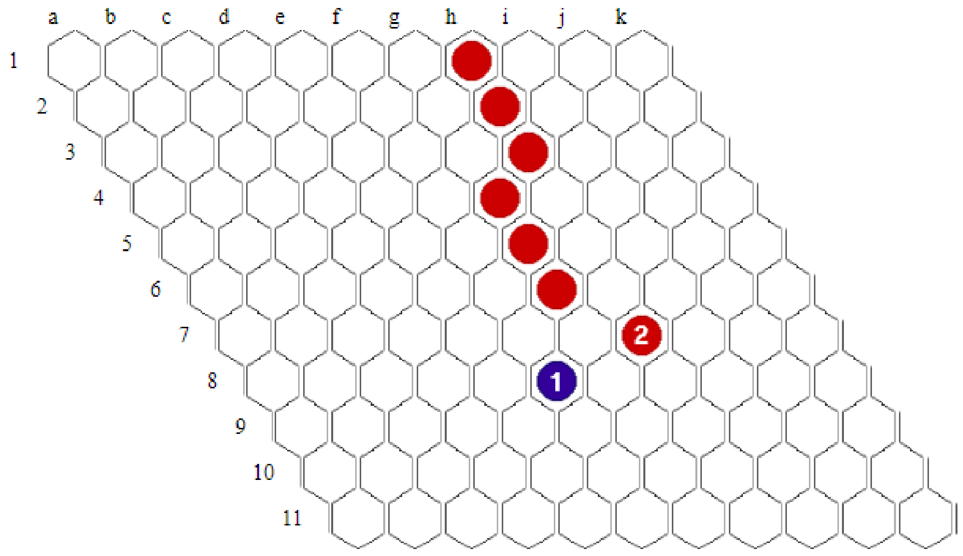
This is called a bridge where two stones of the same colour have been placed with a gap of two in-between the opponent can play in either space but the other player will play in the other which will create a continuous path. There can also just be a simplified bridge called the two bridge where it's just two tiles that have a two gap in-between

Blocking moves is also an integral part of the game. When you have no pieces in the area, it is usually best to start blocking broadly close to at least one of your edges and not too close to the opponent's piece. If your blocking move has too little influence on both your edges then the opponent has more room to manoeuvre there chain around. The most important thing for a player to do is to avoid the mistake of repeatedly trying to block by playing adjacent to the head of the chain. Playing ahead of the chain gives you a move or two to place your pieces before the advancing chain meets your pieces. Thereby eliminating more paths to use.

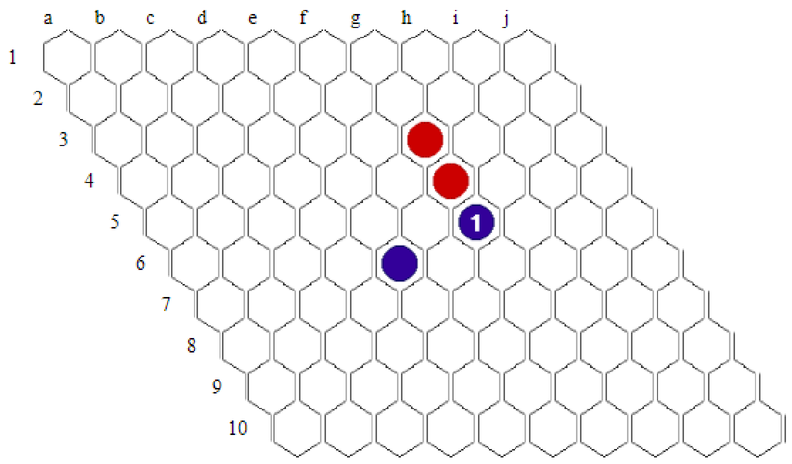
In hex a good offense and a good defence are equivalent as if you can form a connection between your sides then they cannot create their own unbroken path. When playing you will want to focus on your opponents weakest link and reinforce your own weakest link. This is due to the fact that if they can block of your weakest point then your whole chain will become less usable.

Momentum is also a big component of the game as the player that has to respond to his opponents moves won't be able to further his own chain which means that they can't win as easily. Therefore the player with the momentum typically has a large advantage and this advantage normally wins you the game

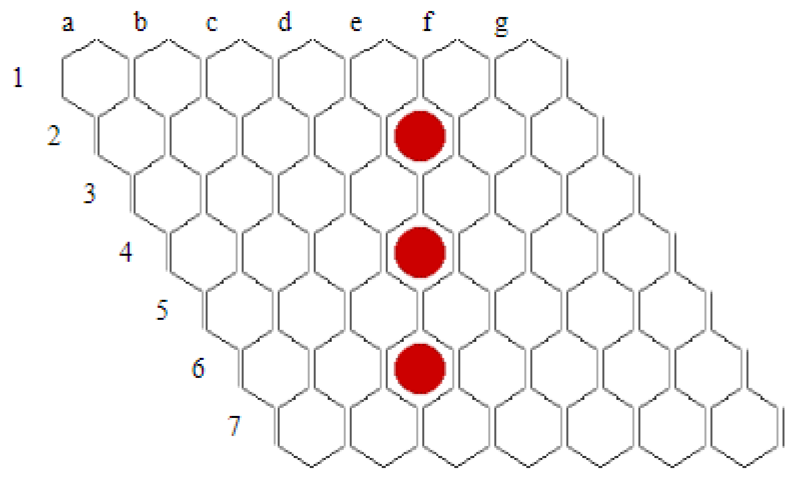
Whenever a player can they should make each of their moves achieve at least two different goals or threats. As a move that creates multiple threats will typically be hard to stop and could end up wining you the game. The central region of the board is strategically the most important area. From the centre, connections can spread out in many directions giving you more flexibility and options than starting from an edge. Furthermore, centrally played pieces are more nearly equidistant from both of your edges which will end up making your weakest links in the chain not as bad.



A bad block as the opponent can still easily manoeuvre his chain around your piece



A good block as the chain can't get around piece 1 as you can block on your turn



Two bridges set up in a row

Type of game

A sequential game is a game where one player chooses their action before the others choose theirs. Hex is an example of a sequential game as

Hex is a game with perfect information. This is where a sequential game has perfect information if each player, when making any decision, is perfectly informed of all the events that have previously occurred. Games with perfect information make the games entirely based on skill as there is no luck in guessing what everyone else has done but relies on you looking at the information and making the best move

As hex has a symmetric board there is no advantage for which side you start on other than the starting player having a slight advantage. However there are ways to make this advantage less impactful on the game

Hex is a strategy game in which the player’s decision-making skills have a high significance in determining the outcome. Almost all strategy games require internal decision tree style thinking, and typically very high situational awareness. It is also an abstract strategy game which is where the theme is not important to the playing experience. Combinatorial games have no randomizers such as dice, no simultaneous movement, nor hidden information.

Theory's and proofs

There are a lot of mathematical proofs and theory's linked to the Hex game

The Hex theorem consists of stating that a game of hex cannot be ended in a draw as connecting and blocking the opponents is an equivalent act

1. Begin with a Hex board completely filled with hexagons marked with either X or O (indicating which player played on that hexagon).
2. Starting at a hexagon vertex at the corner of the board where the X side and O sides meet, draw a path along the edges between hexagons with different X/O markings.
3. Since every vertex of the path is surrounded by three hexagons, the path cannot self-intersect or loop, since the intersecting portion of the path would have to approach between two hexagons of the same marking. So, the path must terminate.
4. The path cannot terminate in the middle of the board since every edge of the path ends in a node surrounded by three hexagons—two of which have to be differently marked by construction. The third hexagon must be differently marked from the two adjacent to the path, so the path can continue to one side or the other of the third hexagon.
5. Similarly, if the sides of the board are considered to be a solid wall of X or O hexagons, depending on which player is trying to connect there, then the path cannot terminate on the sides.
6. Thus the path can only terminate on another corner.
7. The hexagons on either side of the line form an unbroken chain of X hexagons on one side and O hexagons on the other by construction.
8. The path cannot terminate on the opposite corner because the X and O markings would be reversed at that corner, violating the construction rule of the path.
9. Since the path connects adjacent corners, the side of the board between the two corners (say, an X side) is cut off from the rest of the board by an unbroken chain of the opposite markings (O in this case). That unbroken chain necessarily connects the other two sides adjacent to the corners.
10. Thus, the completely filled Hex board must have a winner.

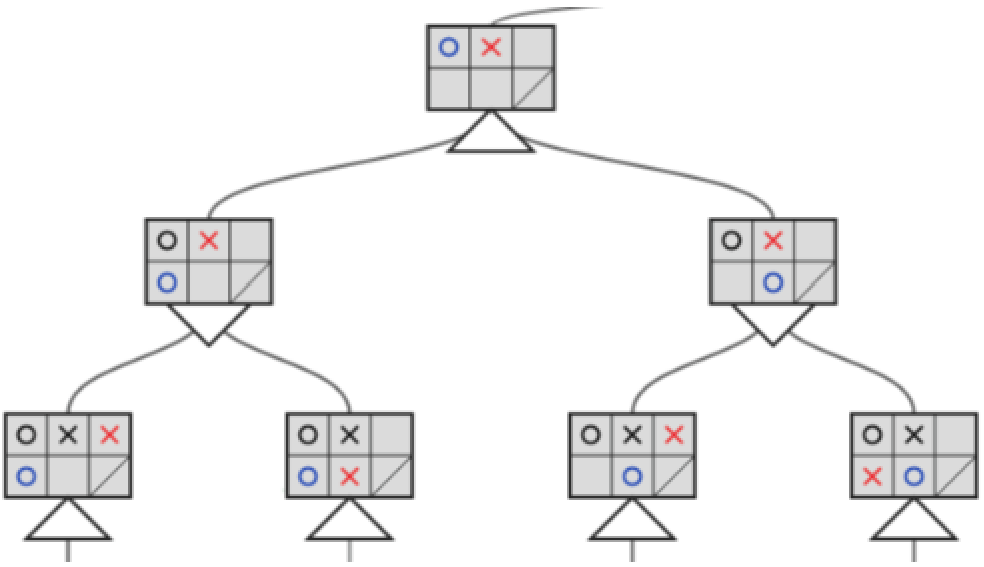
(Taken from Wikipedia the proof that there can't be a draw)

Due to the above proof it was concluded that there must be a winning proof that if you follow it you will always win. IT was also concluded that the first player must have a winning strategy.

1. Either the first or second player must win, therefore there must be a winning strategy for either the first or second player.
2. Let us assume that the second player has a winning strategy.
3. The first player can now adopt the following defense. He makes an arbitrary move. Thereafter he plays the winning second player strategy assumed above. If in playing this strategy, he is required to play on the cell where an arbitrary move was made, he makes another arbitrary move. In this way he plays the winning strategy with one extra piece always on the board.
4. This extra piece cannot interfere with the first player's imitation of the winning strategy, for an extra piece is always an asset and never a handicap. Therefore, the first player can win.
5. Because we have now contradicted our assumption that there is a winning strategy for the second player, we are forced to drop this assumption.
6. Consequently, there must be a winning strategy for the first player.

   (Taken from Wikipedia first player must win)

The winning strategies have been found for smaller board sizes but for larger are still being found. The 7 by 7 grid was found using a decomposition method with a set of reusable local patterns



A decision tree like this will be the simplest way of calculating a few moves ahead to see what the best moves available are. This allows you to look at the future states of the board and decide what moves give you the best chance of winning later on

Interview

I am interviewing Colin Zack who will be the client for the game he would like a way to play Hex at a varying degree of difficulty without having to need another person and would like an easy way to play it without needing to setup and find someone to play with. Having a computerized version of the game will mean that he won't need to worry about setting it up or putting it away, Having a computer to play against will make it so he can play at any time he wants as well.

What difficulty would you like the computer to play with?

I would like the computer to be a able to play a variety of different levels this way I can gradually improve my understanding and techniques of the game. Having different levels of opponents can also make me know which strategies should be used when

What do you want the options for the board to be?

I would like to be able to change the board sizes up to a certain size. This will allow me to play either a quicker game on a shorter board or a longer game on a bigger board. It would be nice to have it so you can choose the colour you play as.

Do you want a two player option as well?

Yes a two player option would be helpful

What sort of rules do you want?

The pie rule would be a must have as it can help balance out the advantage of starting first. But other than that the typical rules will be fine

(The pie rule is where the second player gets two options after the first player has made his move. These options are letting the move stand and switching places. Letting it stand is where the second player remains the second player and moves immediately. Switching places is where the second player becomes the first-moving player, and the "new" second player then makes their "first" move. (I.e., the game proceeds from the opening move already made, with roles reversed.)

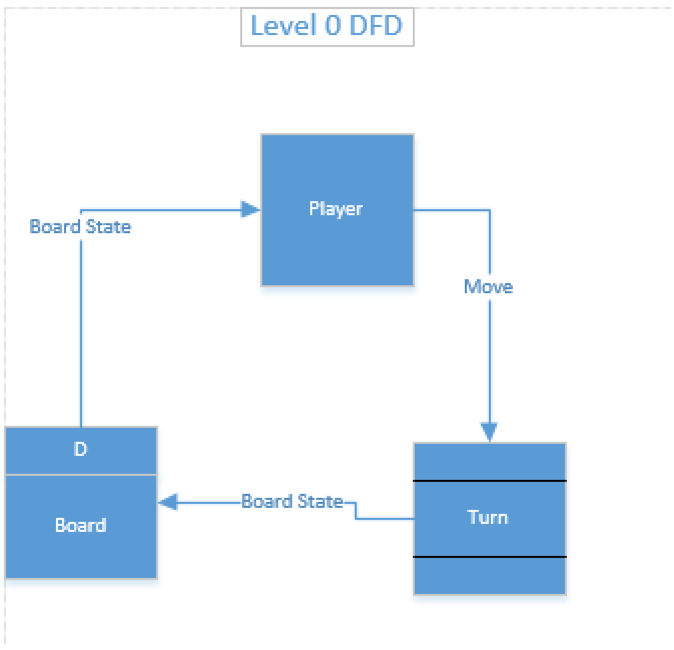
Do you want custom board shapes?

Yes custom Board shapes would be a nice feature but only shapes like a game of Y or a rectangular board would really be needed

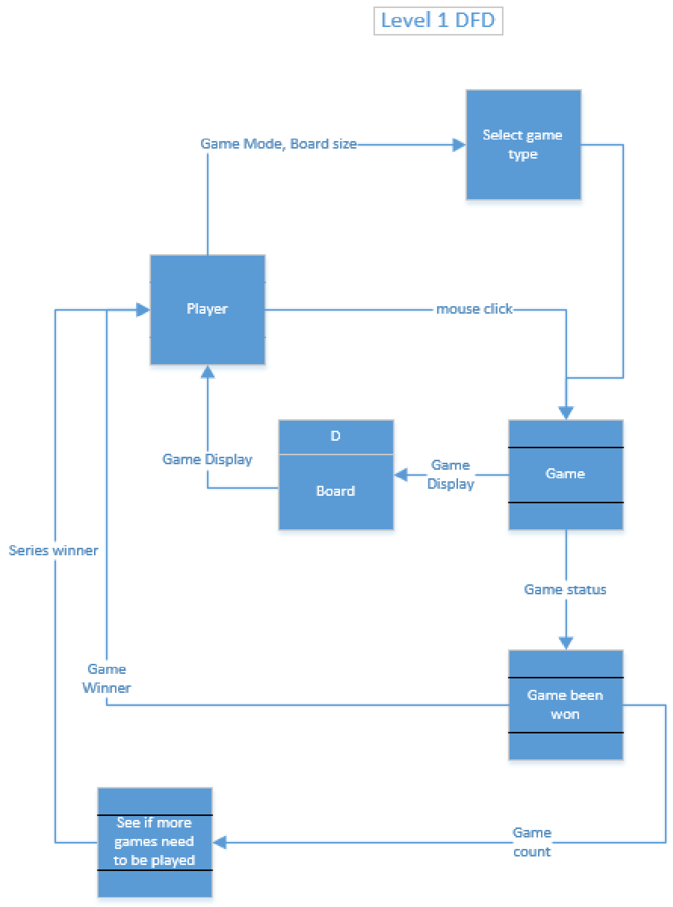
Do you want to know what the optimal moves are when you are playing?

Having an option which shows you the optimal move would be a helpful tool to improve my strategies and help me win the game

Analysis



To the left is a level 0 data flow diagram. It shows the basic flow of data from the player and game. This diagram shows that the player will make a move then process called turn will output the new board state where it is stored as the new board. This board state will then be show to the player where they can then make the decision on move. The flow of data will then continue till a player has won the game. This data flow diagram is a very basic representation of hex with none of the extra features I will add to the game



To the right is a level 1 data flow diagram. It is a much more complex version of the data flow diagram. This one includes the more custom settings for the Hex game I would like to make. This data flow diagram includes the options for game mode and board size. These options will drastically change the amount and type of data that will flow around as a smaller board size will. Different shaped boards will then also have a much different game display as the game display wouldn't be the same. The Player board and game data flow is basically the same but I changed move to mouse click as I will have the player click a button to make their move. Other than that the flow of game display will be the same. There is also more detail added onto what happens when the game has been won. It includes a thing to show the player who has won as this would be an important feature to have as it provides a clear

Level 1 DFD Continued

Message in case the players didn't know who won. There is also a way to have a series of multiple games so that you could compete over a few games if the players want a longer experience. Again it will show the series score and winner at the end to affirm who's won and to remind the players of the series score. These data flow diagrams are designed for two human players playing against each other and not to have any computerized players.

ISPO's

ISPO diagrams are input, process, store and output diagrams. They show what an input will cause and all the different processes that will occur due to that input. It will then show the different outputs that can occur from the output

ISPO For Game turn

Input	Process
• Player Moves (Mouse Click)	• Check colour of player • Colour space of move made • Check if a path has been made • Update current player
Store	**Output**
• Board State • Player Turn • Series score • Has the game been won	• Display board • Display who's turn it is • Display series score

This ISPO tells us that the display will be quite simple but there will be many processes that occur

ISPO for AI

Input	Process
• Player Moves (Mouse Click)	• Gets Board state • Figure out best move for AI • Update board state • Check if game has been won • Update current player
Store	**Output**
• Board State • Player Turn • Series score • Has the game been won	• AI move

ISPO for front end

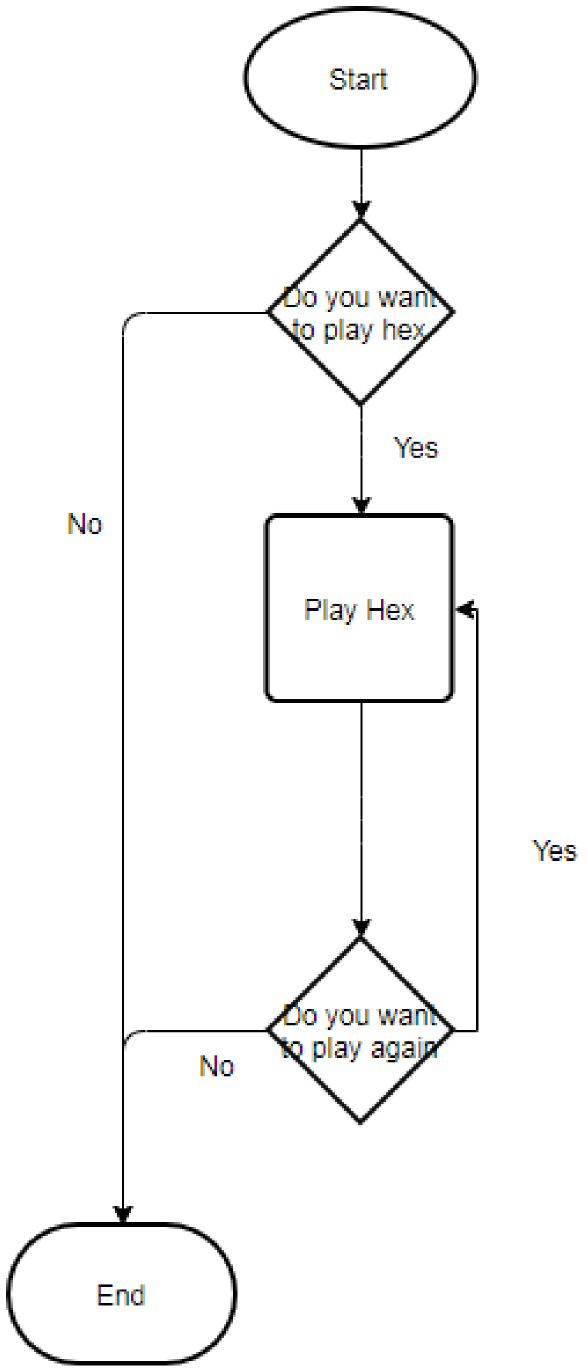
Input	Process
• Clicks on different settings	• Switch between one and two players • Change the difficulty of the AI • Change the size of the boards • Change the shape of the board • Choose the amount of games in the series • Which player goes first
Store	**Output**
• AI difficulty • Board size • Board shape • Amount of games in the series • Who goes first	• Board size changes • Board shape changes • AI difficulty changes • Which player goes first • Amount of games in series

ISPO after series has been played

Input	Process
• Clicks on play or quit	• Reset game score • Reset Game board • Use last games settings • Quit game
Store	**Output**
• Last games settings	• New settings if that option is chosen

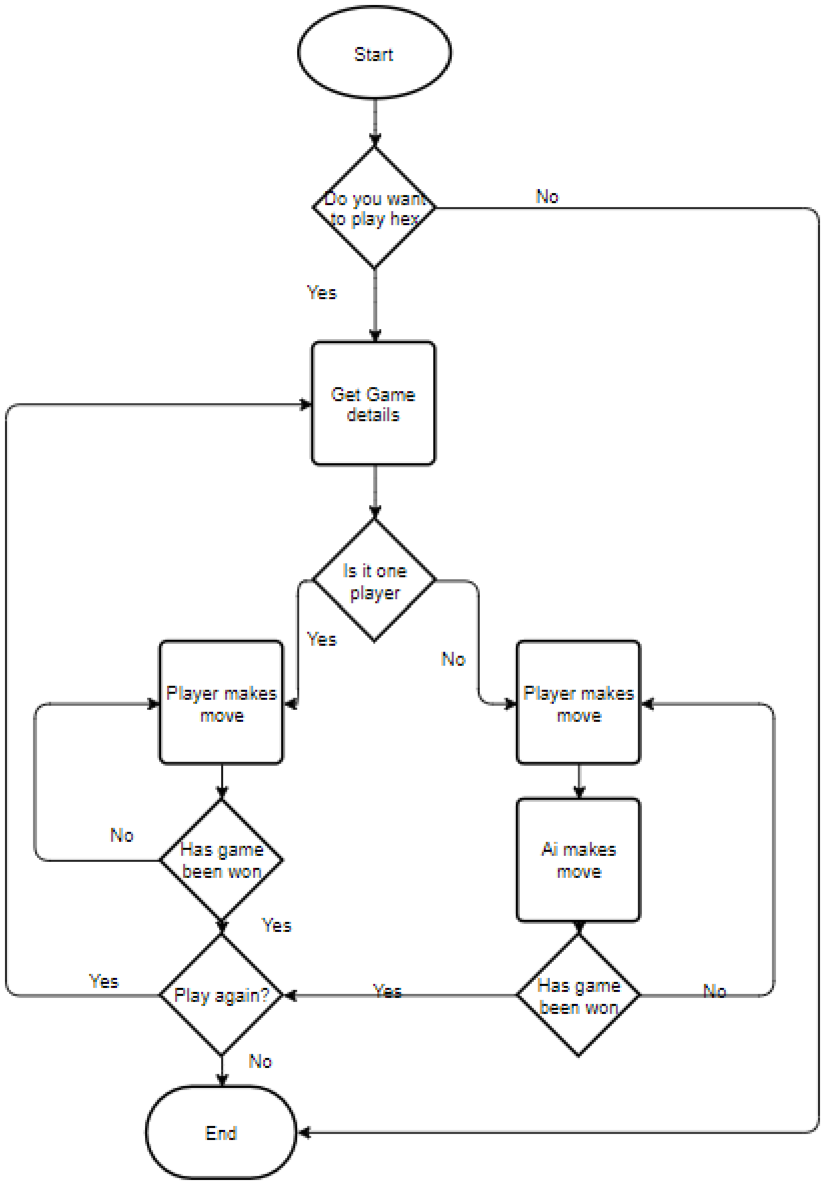
Flowcharts

Flowchart for basic game



This is a basic flowchart for a generic Hex game it has no user choice for different board sizes and board types. It's a very basic version of hex where you just play the game. Complexity is hidden as play hex contains all the actual playing of the game

Flowchart for more complex game



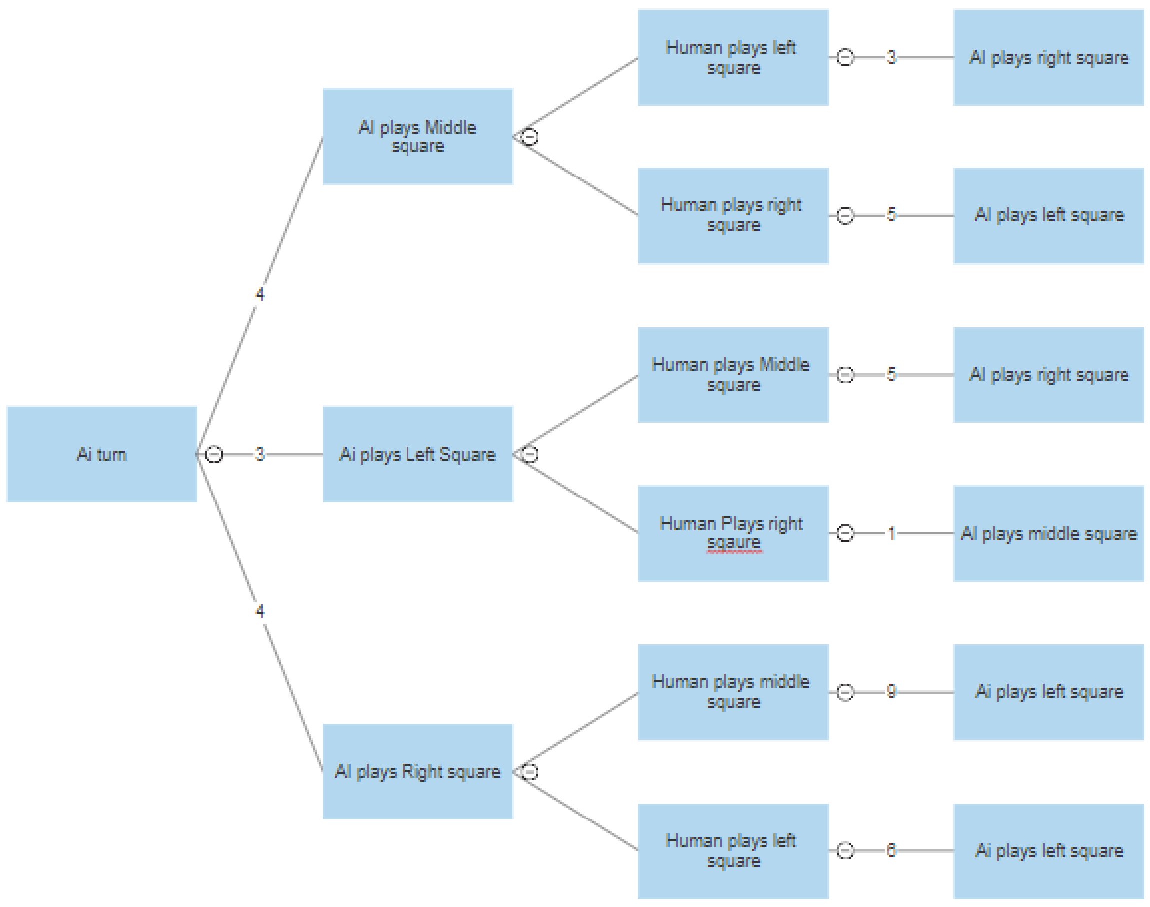
This hex diagram is more specific for my game of hex as it includes an option for an AI player. This allows you to have a single player. This flowchart also contains a process that would allow for different game details. This is what I would do in the coding part where I have a way to get all the different game details such as AI difficulty, board size and board shape. This is a very surface level flowchart that just shows the basic actions of what my hex game would entail. Again such complexities such as how the game has been won is hidden

At the moment there are two different ways that I could have the program play the first is to use MiniMax and the second is to use a shortest path diagram

Game tree diagram

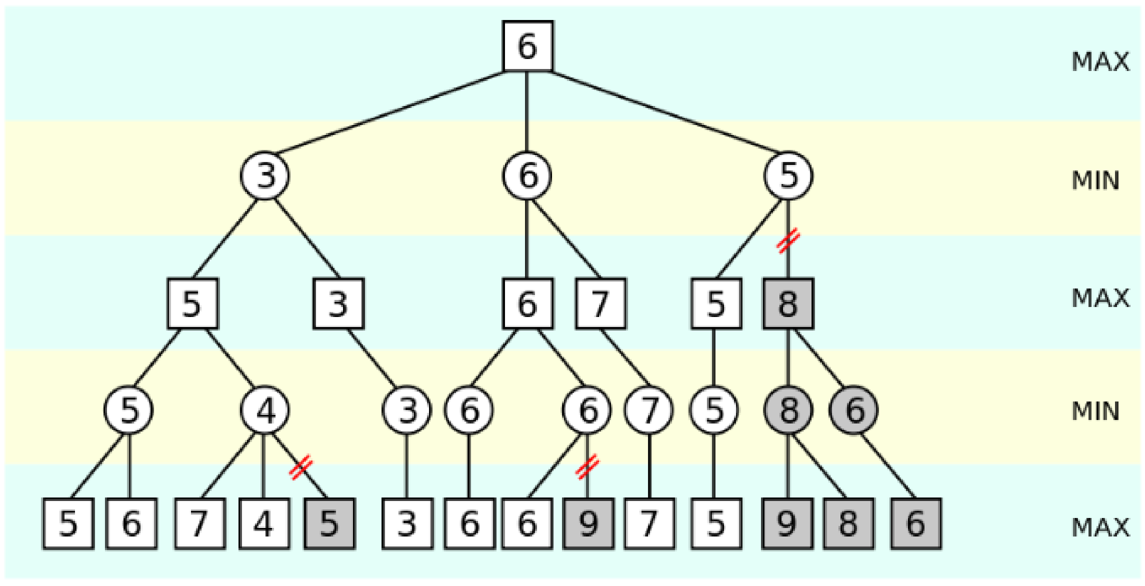
A game tree diagram is a tree diagram that has its nodes as positions in a game and the edges as moves. A complete game tree will contain all possible moves from each position. The number of leafs on a tree diagram represents the total possible number of ways the game can be played. Game trees are important in artificial intelligence because one way to pick the best move in a game is to search the game tree using any of numerous tree search algorithms, combined with minimax-like rules to prune the tree.

When a game has a complete game tree then it is possible to solve the game which means to play it perfectly where either the first or second player will win every time.



The game tree above shows the possible ways the top three tiles of a 3 by 3 hex grid could be played. It is an incomplete game tree as it is missing some of the possible moves. It lists that at first there are three possible moves for the AI then the human can take one of the remaining two options and then the Ai can only choose one remaining option. The numbers on the lines connecting the nodes are an arbitrary number that I've used to show the likelihood of winning. The AI will then choose the path that gives them the highest chance of winning. In the case above I've used random numbers where the higher the number the higher the chance of winning. From this the Ai would always play the right square first and would then want the player to take the middle square. However the difficulty of the AI can be changed by changing how far it looks ahead in the tree diagram. As if in the one above the AI only looked at the outcome of the next move then it would play middle square half the time and would play the right square half the time.

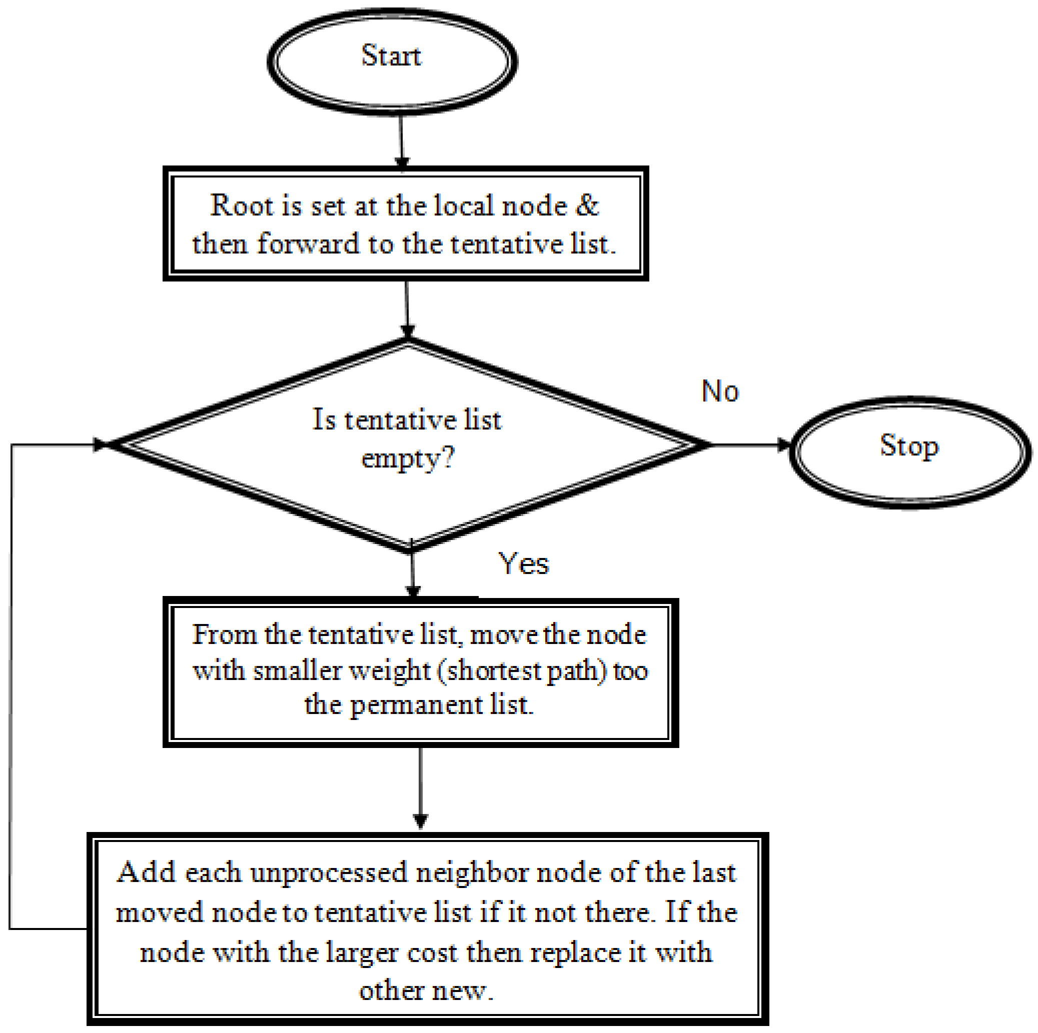
However if the Ai was going second in the diagram above the AI would want to find the smallest values connecting the nodes as that makes it have a higher chance of winning. This means that the game tree diagram would be different depending on whether the AI will play first or second



α-β Pruning

Alpha beta pruning on a two person game tree of 4 plies

Dijkstra's Diagram


Start
Root is set at the local node & then forward to the tentative list.
Is tentative list empty?
No
Stop
Yes
From the tentative list, move the node with smaller weight (shortest path) too the permanent list.
Add each unprocessed neighbor node of the last moved node to tentative list if it not there. If the node with the larger cost then replace it with other new.

This diagram shows how a Dijkstra search would occur. This flowchart says that the start of the search starts at one unique point then searches all the possible points around it. From all the possible points that are around it the one with the smallest weight is chosen. Then the next closest point is found out and the length is the two points added together if this new paths shortest point comes off of the point that was found. This search will keep happening until either all the points are found or the shortest distance to each is found or until the point you want to reach is found.

In my game the weightings of the paths would be determined on whether they are already clicked by a player or not. A tile clicked by your opponent would have the highest weighting and a tile clicked by yourself would have the lowest weighting. In hex the Dijkstra's would try to get from either left to right or top to bottom in the least amount of moves.

Data Dictionary

A set of information describing the contents, format, and structure of a database and the relationship between its elements, used to control access to and manipulation of the database.

Data dictionary for set up of game

Name:	Data Type:	Length:	Purpose:
SeriesLength	integer		This is used to hold how many games the series will be made up of
BoardSize	Integer		This is used to draw the board by creating the right amount of buttons for length and width
BoardShape	string		This is where you choose from some premade shapes for the board
AI difficulty	Integer	1-?	This is where the difficulty of the ai will be stored so that the Ai will know how far to look on the tree diagram
StartingPlayer	Char	R or B	This decides what player is set as the current player at the start of the game
NumberofPlayer	Integer	1-2	This holds how many players will be playing which determines if an AI is needed
PiRule	boolean		This will hold whether the player has chosen to play using the Pi rule or just normally

Data dictionary for Playing game

Name:	Data Type:	Length:	Purpose:
CurrentPlayer	Character		This is where the player is represented by the starting letter of their colour so you know what player is going
ButtonsPressed	Boolean		This holds whether a button has been pressed so can't be pressed again
GameWon	boolean		This holds whether a game has been won or if it needs to continue
SeriesScore	Integer		This holds the score of the series of games so far

Data Dictionary for playing again

Name:	Data Type:	Length:	Purpose:
Playagain	Boolean		This holds whether the players want to play again
FinalScore	Integer		This holds the final score of the series and will show who won
SettingsBefore			This holds all the settings of the previous series

Requirements

Dialogue

One of the first things I was thinking about with the hex game was how to create and AI player for the game. I decided to make it have a range of difficulty options where the computer would play at varying degrees of efficiency. My first problem with creating an ai was on how to actually create one. I had two different options Minimax and shortest path algorithm. In the end I decided to use a shortest path algorithm. After researching different path finding algorithms I decided I would use a Dijkstra's algorithm this is because I could simply turn the board into a undirected graph with weights. This path finding algorithm would also be able to work on all sizes of the board which would allow me to not have to create multiple different algorithms to make the computer play. The next thing I would have to figure out after choosing a path finding algorithm was how to have different difficulty levels. After a while I realised that I could do the shortest path algorithm for both players as the game is about creating paths and blocking paths. This means that for the highest difficulty I could have a shortest path algorithm run for both players as this will allow you to find the combinatory best move of blocking your opponent and helping your path. While lower difficulties could only look at finding the best move to progress your path

Another thing I had to decide was how to display the board and get the inputs from the user. At first I was contemplating between using the console and VB forms. I decided against using the console as I feel like it would have detracted from the game by making it slower and having a less aesthetically pleasing display. I decided to use VB forms where it will have buttons that could be pressed. This allows much better interaction with the game as you can easily tell what space you are going without having to search across on a grid. I also feel like having buttons being pressed could speed up the play of the game which would make it more enjoyable. Buttons are also very easy to change the colour of when they have been clicked which makes a great visual representation of the different moves. However the problem with buttons is that they don't have a prebuilt hexagon shaped button and only have squares or rectangles. In the end I was able to get hexagon shaped buttons by finding a command that switched the number of sides.

After deciding to use VB forms I decided to use a range of buttons and labels for my front end. The use of labels and buttons allows easy reading of the different options. The use of buttons also makes it easy to tell which options you have chosen and would mean that there could be no options that weren't right from a typo. I then decided to only have the options be there when the game isn't being played this means that I put all the options in the middle of the form so that they would be seen much easier. I then decided to have the options chosen to be displayed as small labels to the side of the game so that you can see when the games loaded that you chose the options that you wanted to

I had to figure out how to determine if someone had won the game or not. My first thoughts at this was checking all the different buttons and seeing whether the colours matched. However this would result in my code being extremely long to compensate for the buttons that I must call different names as to not get them mixed up. However this would have made determining the paths that the buttons take quite simple. The next idea I had was to determine the colour of certain spots on the board to check what colour those buttons are. However I couldn't get it to work and if I had it would make it so I had to have different checks depending on the size of the board. My last idea was that I would create a list of the different rows of buttons and when one was pressed the value of it in the list would change to the colour of the player who pressed it. This way I only needed to determine which button was being pressed by finding the location of the press and then assigning the colour when it's been pressed. After this it would then have a check where I would compare the different layers of the array. I decided to use the last idea as it would make it so my method of creating buttons would still work and the lists would be very easy to compare between

The last few things that I needed to decide on I could just try out on the form itself and decide which I liked best. I looked at different button sizes and decided on quite big buttons so that the game is more of the focus of the screen. I also looked at different colours of the buttons and chose the original blue and red colours that the game typically comes with. I also looked at the spacing between the buttons and made it so they had gaps between so you can more easily tell where the buttons are separated.

Requirements

1. Game is launched
    - 1.1. Options are displayed
        - 1.1.1. Board size
        - 1.1.2. AI difficulty
        - 1.1.3. Number of games
        - 1.1.4. Who goes first
        - 1.1.5. Use Pi rule
        - 1.1.6. Load Game
    - 1.2. User can press options
    - 1.3. Buttons change colour when they are chosen
2. Hex board is created
    - 2.1. Board is the shape that the user picked
    - 2.2. Board is the size the user picked
    - 2.3. Options that were chosen are displayed on the side
3. Player chosen to go first goes
    - 3.1. Button changes colour to their colour
        - 3.1.1. The button pressed changes colour to the current player
4. Second player is asked about pie rule
    - 4.1. They let the play stand
    - 4.2. They switch goes with the original player
5. Second player now goes
6. Game goes until someone wins
7. Has different difficulties
    - 7.1. Difficulty one is the easiest difficulty to beat- Moves chosen at random
    - 7.2. Difficulty two uses Dijkstra's for the current player to find the shortest path
        - 7.2.1. Runs Dijkstra for yourself
        - 7.2.2. Chooses a move which is on the shortest path
    - 7.3. Difficulty three uses a Dijkstra for your opponent and yourself

7.3.1. Runs Dijkstra for yourself
7.3.2. Runs Dijkstra for opponent
7.3.3. Also considers which pieces are around it in order to prioritize making bridges
7.3.4. Combines all three to form a move which best blocks your opponent, creates a bridge and helps complete your path
7.3.5. Chooses the move that does all the things above

8. Player Chooses save game
    8.1. Allows User to choose a Name to save the game as
        8.1.1. Saves the current player
        8.1.2. Saves The board size, difficulty of computer, Number of players, tiles already chosen
9. Load Game
    9.1. Game is loaded with all the relevant information
        Uses the name of the file to load the game
10. Game checks if a player has won
    10.1. Looks at button that was just pressed
        10.1.1. If it has any of the same colour next to it checks the buttons around that one
        10.1.2. Goes through the lists to see if there is a straight path through
    10.2. If the game hasn't been won then players keep making moves until it has
    10.3. When game has been won displays who won
11. If there are more games in the series
    11.1. Repeat steps from 2 to 7
    11.2. Once the series has been won display the score and winner
12. Ask the player if they want to play again
    12.1. If they say no exit the game
    12.2. If they say yes give options
        12.2.1. They can either play with same settings
        12.2.2. Or take them back to step 1

Design

Algorithms used

One of the big algorithms I used in my project is the one used to check whether a player has won the game of Hex or not. This is a very important algorithm as it will determine whether the game will work properly or not. This is because if the game doesn't actually tell you who has won correctly then the game wouldn't work properly. When I was figuring out how to do make the check work properly I decided to use recursion to figure out whether someone had won or not this is because there could be multiple paths that you could take from one tile clicked. This means with recursion you could take those multiple paths quite easily

Pseudocode of algorithm for board size of 5

```
WinCheck(i)

Character→Integer

Whichlist→String(Boardsize)

NextList→String(Boardsize

For I = 1 to BoardSize

          If Character = 1 Then

        Whichlist = ButtonListA

        Nextlist = ButtonlistB

     ElseIf Character = 2 Then

        Whichlist = ButtonlistB

        Nextlist = ButtonlistC

     ElseIf Character = 3 Then

        Whichlist = ButtonlistC

        Nextlist = ButtonlistD

     ElseIf Character = 4 Then

        Whichlist = ButtonlistD

        Nextlist = ButtonlistF

     End If

     Character = Character + 1

If CurrentPlayer = "R" Then

        If Whichlist(i) = "R" Then


          If Whichlist(i) = Nextlist(i - 1) Then
```

```
            If Character = 5 Then
                MessageBox.Show("Red Wins")
                CurrentPlayer = "N"
                Character = 1
            End If
            WinCheck(i - 1)

        End If

        If Whichlist(i) = Nextlist(i) Then
            If Character = 5 Then
                MessageBox.Show("Red wins")
                Character = 1
                CurrentPlayer = "N"
            End If
            WinCheck(i)
        End If
       If i <> BoardSize Then

            If Whichlist(i) = Whichlist(i + 1) And Whichlist(i) = "R" Then
                Character = Character - 1
                WinCheck(i + 1)
            End If
            If Whichlist(i) = Whichlist(i - 1) And Whichlist(i) = "R" Then
                Character = Character - 1
                Checkleft(i - 1)

            End If
        ElseIf Whichlist(i) = Whichlist(i - 1) And Whichlist(i) = "R" Then
            Character = Character - 1
            WinCheck(i - 1)
        End If
      End If
```

This algorithm shows that there are multiple paths that you can check along which is what you want to happen as I stated earlier that many paths can be taken to win. It utilizes checking different lists and comparing the values given in the lists. This works by assigning such values when the buttons are pressed. It will then check the equivalent of the surrounding buttons of the other lists and see if they too are the same colour as then that will be part of the path the path to win can take. For the blue player the algorithm is the same as I made it so they had an equivalent list where it to can use these checks to determine the different routes.

At the beginning I assign the two lists I will compare as two of the lists that I have determined the values of earlier when the button was actually pressed. This means that there will be less checks as I can just use the same check to compare the two lists as the values will be changed by having a different value of "Character". This algorithm is called after a button press this means that every button pressed will cause this algorithm to have at least one pass through. If there has been a winner I declare the current player as N to make it so that the algorithm won't run anymore as this will lead to less passes through as it will know that there are no more checks that need to be made as there is already a path that has been completed

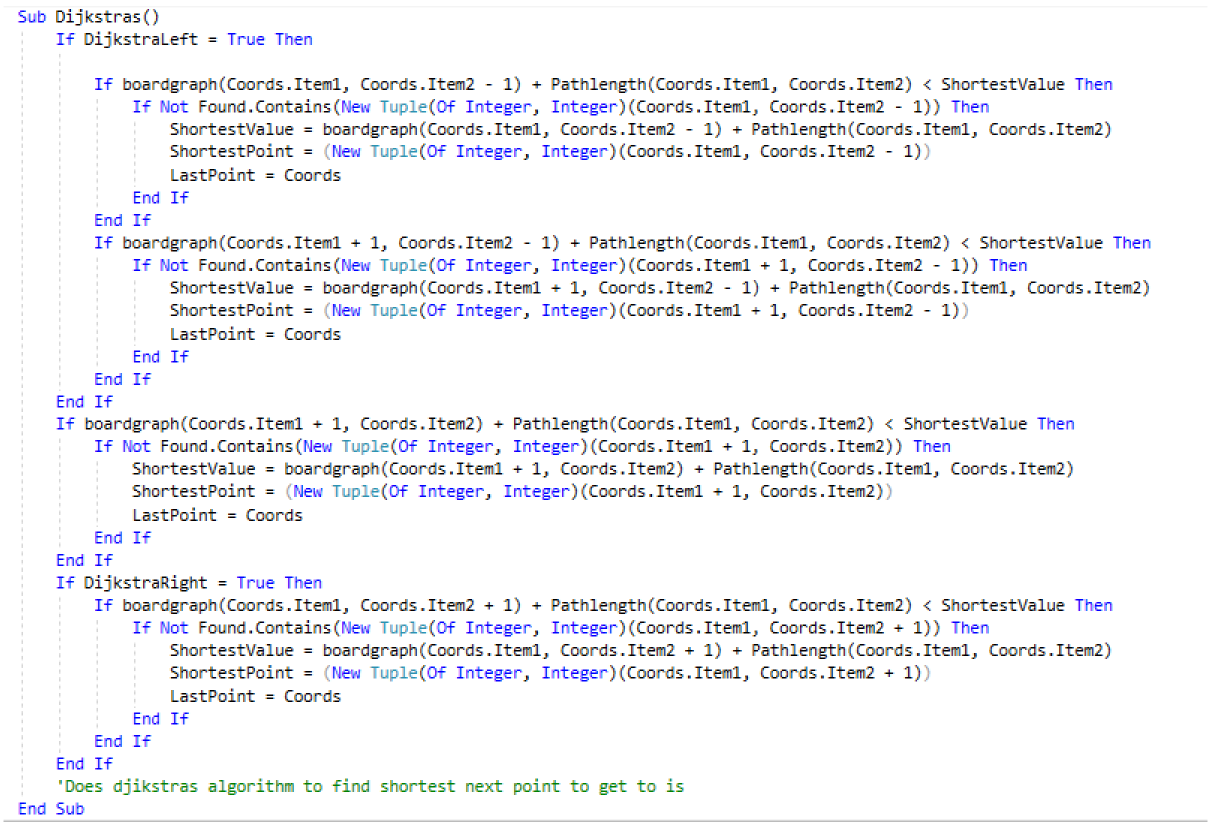
The validation I used was the if i<>Boardsize then, this is because VB can't check a value of an array greater than the upper limit of the arrays which would cause an error to occur. However you still will want it to check the value to the left in that arrays. I used a lot of arrays in my pseudocode as that was the best way for me to visualize the board as each array could represent a column or a row and the value assigned could be what colour that tile was. I also used a few integers the I is to determine how far along the array you go so specifically which button you are looking at while the character is meant to represent the row you are on if you are the red player and the column you are on if you are the blue player. I also check at the beginning what the current player is and check that the value of the buttons around are red.

Pseudocode for generic Dijkstra's search

```
 function Dijkstra(Graph, source):
     create vertex set Q
      for each vertex v in Graph:
         dist[v] ← INFINITY
          prev[v] ← UNDEFINED
```

```
        add v to Q
     dist[source] ← 0
        while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u:           // only v that are still in Q
          alt ← dist[u] + length(u, v)
         if alt < dist[v]:
             dist[v] ← alt
        prev[v] ← u
  return dist[], prev[]
```

This algorithm is very similar to the one I will use as mine will find the shortest distance but each node will be represented by a set of coordinates. The algorithm works by considering all the nodes of the unvisited neighbours and calculates their tentative distance to the current distance and if the distance is smaller than the old distance then replace it with the tentative distance. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbour B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept. Once all the neighbours of the current node are visited then the node is considered visited and won't be checked again. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity then stop. The algorithm has finished. Otherwise, select the unvisited node that is marked with the smallest tentative distance set it as the new current node and repeat looking thorough the surrounding nodes



```vb
Sub Dijkstras()
    If DijkstraLeft = True Then

        If boardgraph(Coords.Item1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2) < ShortestValue Then
            If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 - 1)) Then
                ShortestValue = boardgraph(Coords.Item1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2)
                ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 - 1))
                LastPoint = Coords
            End If
        End If
        If boardgraph(Coords.Item1 + 1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2) < ShortestValue Then
            If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2 - 1)) Then
                ShortestValue = boardgraph(Coords.Item1 + 1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2)
                ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2 - 1))
                LastPoint = Coords
            End If
        End If
    End If
    If boardgraph(Coords.Item1 + 1, Coords.Item2) + Pathlength(Coords.Item1, Coords.Item2) < ShortestValue Then
        If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2)) Then
            ShortestValue = boardgraph(Coords.Item1 + 1, Coords.Item2) + Pathlength(Coords.Item1, Coords.Item2)
            ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2))
            LastPoint = Coords
        End If
    End If
    If DijkstraRight = True Then
        If boardgraph(Coords.Item1, Coords.Item2 + 1) + Pathlength(Coords.Item1, Coords.Item2) < ShortestValue Then
            If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 + 1)) Then
                ShortestValue = boardgraph(Coords.Item1, Coords.Item2 + 1) + Pathlength(Coords.Item1, Coords.Item2)
                ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 + 1))
                LastPoint = Coords
            End If
        End If
    End If
    'Does djikstras algorithm to find shortest next point to get to is
End Sub
```

This is my actual djikstras code.

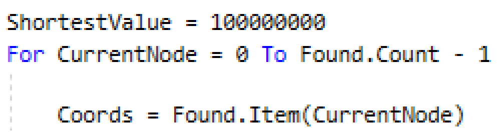
This checks the coordinates around all the point that was passed in. It has four different checks for around the point to the left of it, to the right of it and down left and right from it. However not all coordinates have tiles in all these places so this will check only the places that a tile is at(outside the range of the board) . This is why it has Dijkstra right and left as Boolean as when these are true then the code knows that there is a tile to the left/ right of the current tile.

This code works by comparing the shortest value set so far (on the first point it is set as a number higher than anything you could get this way the first valid move will become the shortest path straight away. This shortest value is then compared to the path length of the current node plus the distance to the node it is looking at. This way you can get the full path length of the surrounding nodes

The validation for this code is to make it so that the code doesn't loop infinitely by always just going to the same point this is stopped by the IF NOT statement. This works by looking at the coordinate that was just found out to be the shortest distance away so far. This coordinate is then compared to a list of coordinates that is called found. This found contains all the coordinates that have been visited so far by the algorithm. If the coordinate that was currently just looked at is in the list then it doesn't change the value of shortest value and doesn't make the shortest value part of that list . This means that this coordinate won't be put in found again.

After it has been checked that the point isn't in the list of coordinates that have already looked at then it will change the shortest value to the value of the path length up to this new point. Next it sets the shortest point equal to the point that it is looking at this is so that the coordinates of the point is saved so that at the end you can add it to the list of found coordinates and search from that point in future loops of the algorithm if this ends up being the shortest point.

The last point is the tile which has been used to get to the shortest path. This saves it so that at the end of looking around all the found points you know which found point was used to get to the shortest point away.



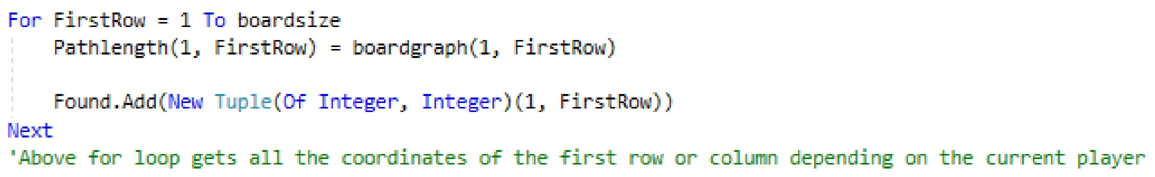
```
ShortestValue = 100000000
For CurrentNode = 0 To Found.Count - 1

    Coords = Found.Item(CurrentNode)
```

This is the loop used to look at all the nodes that are in the found list

This means that the Dijkstra's will run for the amount of Nodes that are in the list of found coordinates. Coords is set to the current nodes coordinates so that when you run the Dijkstra's algorithm you have the cords you are currently looking at. Coords is set as a tuple as you can then split it into its x and y components (row and column) and use those values to look at the coordinates around that point.

This For loop will run multiple times as each step through Found will increase by 1 until the whole board has been found. This way as it steps through each coordinate that has been found so far it is able to search through all the surroundings of all the coordinates

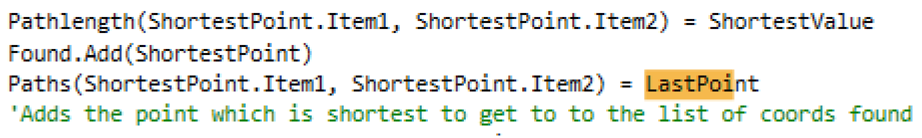
Shortest value is reset each loop on the outside so that the shortest point doesn't stay as shortest point of the first run through of dijkstras



```
For FirstRow = 1 To boardsize
    Pathlength(1, FirstRow) = boardgraph(1, FirstRow)

    Found.Add(New Tuple(Of Integer, Integer)(1, FirstRow))
Next
'Above for loop gets all the coordinates of the first row or column depending on the current player
```

This is the code that gets all the path lengths of the first row. This way the Dijkstra's will have a start point to look from.

This looks at the whole of the first row for the current player and gets the value of those coordinates it then adds the points to the found list so that it can search around all those points. This works for all any board size because of the for loop uses the value of board size.

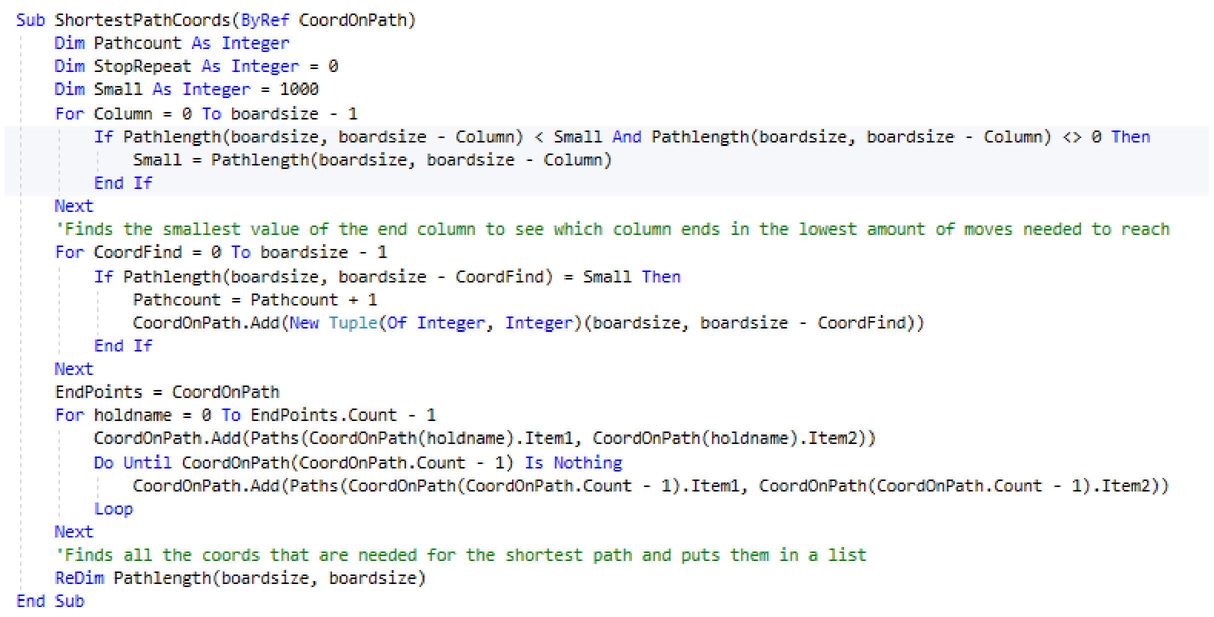


```
Pathlength(ShortestPoint.Item1, ShortestPoint.Item2) = ShortestValue
Found.Add(ShortestPoint)
Paths(ShortestPoint.Item1, ShortestPoint.Item2) = LastPoint
'Adds the point which is shortest to get to to the list of coords found
```

This is where the path length of the point that was found to be the shortest away from the first row of points is added to the path length array so that all the points that come from this can use this value without recalculating it again.

This is also where the other coordinates are added to the found list.

The Paths list holds the coordinates of the point that was used to get to that point. This way you can transverse the path backwards to find all the coordinates that lie on it



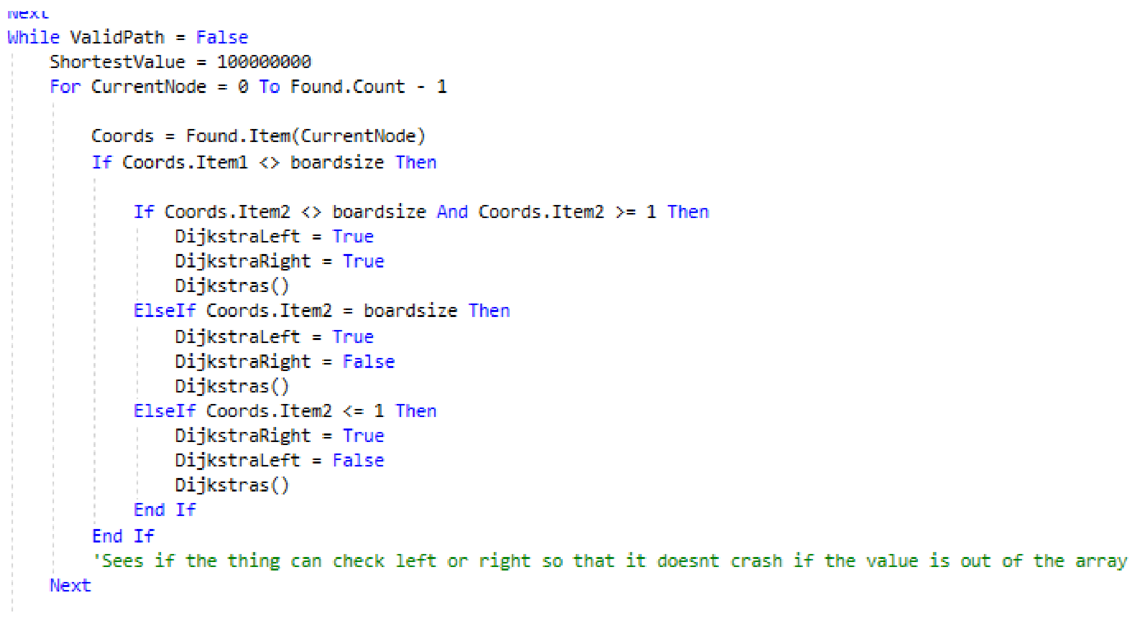
```
Sub ShortestPathCoords(ByRef CoordOnPath)
    Dim Pathcount As Integer
    Dim StopRepeat As Integer = 0
    Dim Small As Integer = 1000
    For Column = 0 To boardsize - 1
        If Pathlength(boardsize, boardsize - Column) < Small And Pathlength(boardsize, boardsize - Column) <> 0 Then
            Small = Pathlength(boardsize, boardsize - Column)
        End If
    Next
    'Finds the smallest value of the end column to see which column ends in the lowest amount of moves needed to reach
    For CoordFind = 0 To boardsize - 1
        If Pathlength(boardsize, boardsize - CoordFind) = Small Then
            Pathcount = Pathcount + 1
            CoordOnPath.Add(New Tuple(Of Integer, Integer)(boardsize, boardsize - CoordFind))
        End If
    Next
    EndPoints = CoordOnPath
    For holdname = 0 To EndPoints.Count - 1
        CoordOnPath.Add(Paths(CoordOnPath(holdname).Item1, CoordOnPath(holdname).Item2))
        Do Until CoordOnPath(CoordOnPath.Count - 1) Is Nothing
            CoordOnPath.Add(Paths(CoordOnPath(CoordOnPath.Count - 1).Item1, CoordOnPath(CoordOnPath.Count - 1).Item2))
        Loop
    Next
    'Finds all the coords that are needed for the shortest path and puts them in a list
    ReDim Pathlength(boardsize, boardsize)
End Sub
```

This piece of code is called after Dijkstra's has been run and all the path lengths of each individual tile is found

This is used to determine from all the path lengths that were gathered which path is he shortest out of them all. First the code runs through all of the last row/ column of the path length array and finds the lowest out of that list. There is some code to stop the program from having an out of range error by having it so that the last row can't have a path length of 0 as this would mean that a player would have already won. This stops the code from returning the path of the 0 index as my code is indexed at 1

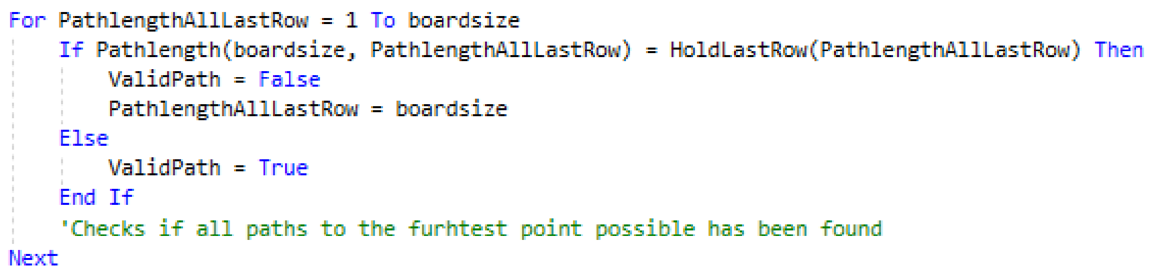
Next the ShortestPathCoords sub uses a for loop in order to find out if more than one point on the last row of buttons has the shortest path length. This is done as multiple points can have the same path length and if they do and the path length is the shortest then you would want to map out the multiple paths as they will have the same level of quality of move. These points are all added to a list which contains there coordinates

After this the sub uses the array paths which contains the node that was used to get to the node you currently are on. This is stuck on a for loop so that you can reuse this code so that it can be used for all of the endpoints. It will then transverse down this path to get all the coordinates that lie on the path again using paths to find the tile that was used to get to it. The nodes which are found to be on the shortest path are all added to a list of coordinates (tuples). This list will contain the complete path of all the coordinates that lie on all the different paths



```
Next
While ValidPath = False
    ShortestValue = 100000000
    For CurrentNode = 0 To Found.Count - 1

        Coords = Found.Item(CurrentNode)
        If Coords.Item1 <> boardsize Then

            If Coords.Item2 <> boardsize And Coords.Item2 >= 1 Then
                DijkstraLeft = True
                DijkstraRight = True
                Dijkstras()
            ElseIf Coords.Item2 = boardsize Then
                DijkstraLeft = True
                DijkstraRight = False
                Dijkstras()
            ElseIf Coords.Item2 <= 1 Then
                DijkstraRight = True
                DijkstraLeft = False
                Dijkstras()
            End If
        End If
        'Sees if the thing can check left or right so that it doesnt crash if the value is out of the array
    Next
```

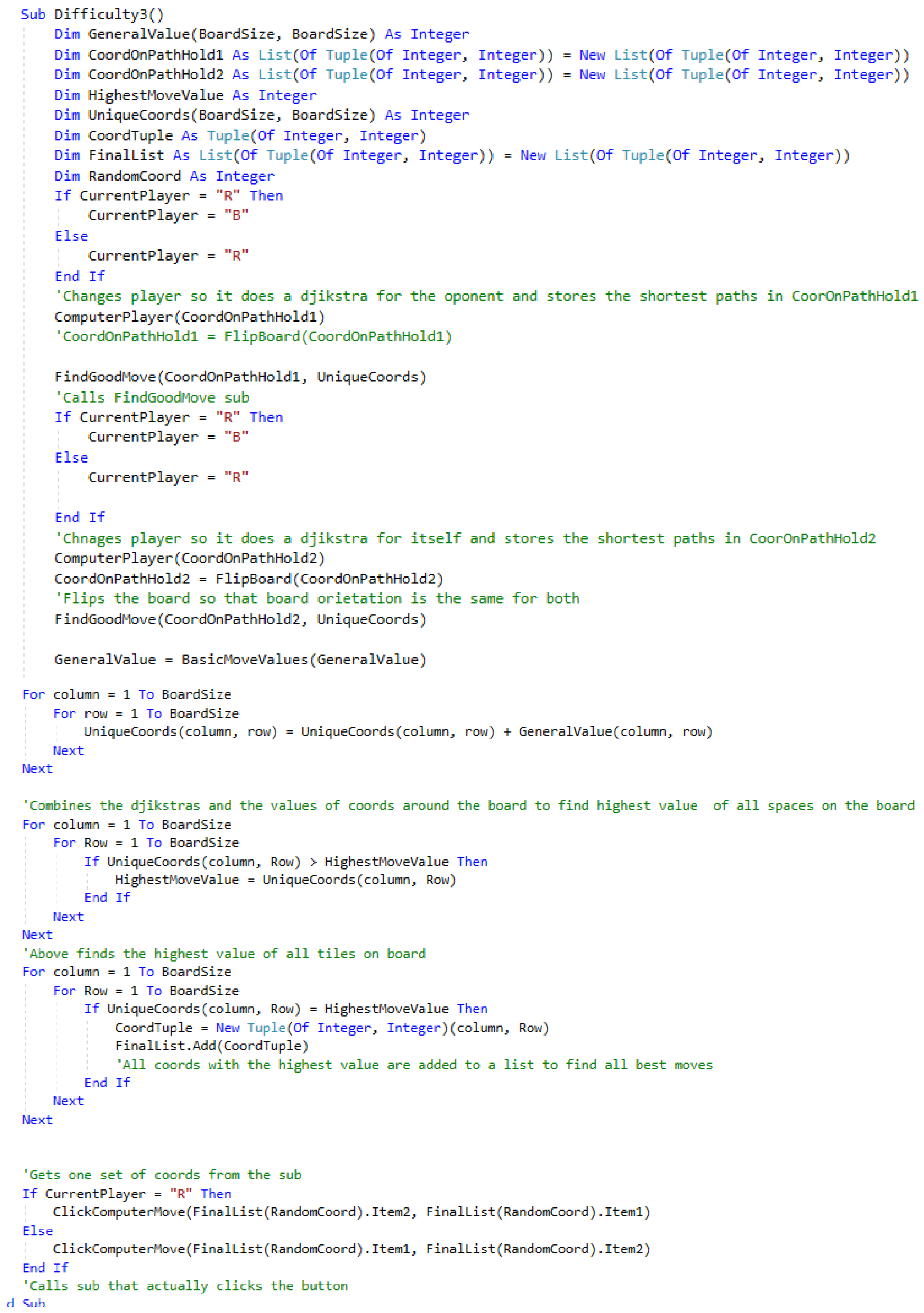
Small amount of validation to stop an index out or range occurring this is because the algorithm would try and check the spaces that don't exit next to the edges of the board



```
For PathlengthAllLastRow = 1 To boardsize
    If Pathlength(boardsize, PathlengthAllLastRow) = HoldLastRow(PathlengthAllLastRow) Then
        ValidPath = False
        PathlengthAllLastRow = boardsize
    Else
        ValidPath = True
    End If
    'Checks if all paths to the furhtest point possible has been found
Next
```

This is used to see if the shortest path to the whole last row has been found. This is useful especially for difficulty 3 as it is a combination of two different Dijkstra's and the addition of a general value this means that you need to find the path values for the whole board in order to do this.

This works by having a value for each end row at the start of the Dijkstra's and if the value for the path length of the last row is still equal to the value it was given at the start then the Dijkstra's algorithm will continue running



```
Sub Difficulty3()
    Dim GeneralValue(BoardSize, BoardSize) As Integer
    Dim CoordOnPathHold1 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
    Dim CoordOnPathHold2 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
    Dim HighestMoveValue As Integer
    Dim UniqueCoords(BoardSize, BoardSize) As Integer
    Dim CoordTuple As Tuple(Of Integer, Integer)
    Dim FinalList As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
    Dim RandomCoord As Integer
    If CurrentPlayer = "R" Then
        CurrentPlayer = "B"
    Else
        CurrentPlayer = "R"
    End If
    'Changes player so it does a djikstra for the oponent and stores the shortest paths in CoorOnPathHold1
    ComputerPlayer(CoordOnPathHold1)
    'CoordOnPathHold1 = FlipBoard(CoordOnPathHold1)

    FindGoodMove(CoordOnPathHold1, UniqueCoords)
    'Calls FindGoodMove sub
    If CurrentPlayer = "R" Then
        CurrentPlayer = "B"
    Else
        CurrentPlayer = "R"

    End If
    'Chnages player so it does a djikstra for itself and stores the shortest paths in CoorOnPathHold2
    ComputerPlayer(CoordOnPathHold2)
    CoordOnPathHold2 = FlipBoard(CoordOnPathHold2)
    'Flips the board so that board orietation is the same for both
    FindGoodMove(CoordOnPathHold2, UniqueCoords)

    GeneralValue = BasicMoveValues(GeneralValue)

For column = 1 To BoardSize
    For row = 1 To BoardSize
        UniqueCoords(column, row) = UniqueCoords(column, row) + GeneralValue(column, row)
    Next
Next

'Combines the djikstras and the values of coords around the board to find highest value  of all spaces on the board
For column = 1 To BoardSize
    For Row = 1 To BoardSize
        If UniqueCoords(column, Row) > HighestMoveValue Then
            HighestMoveValue = UniqueCoords(column, Row)
        End If
    Next
Next
'Above finds the highest value of all tiles on board
For column = 1 To BoardSize
    For Row = 1 To BoardSize
        If UniqueCoords(column, Row) = HighestMoveValue Then
            CoordTuple = New Tuple(Of Integer, Integer)(column, Row)
            FinalList.Add(CoordTuple)
            'All coords with the highest value are added to a list to find all best moves
        End If
    Next
Next


'Gets one set of coords from the sub
If CurrentPlayer = "R" Then
    ClickComputerMove(FinalList(RandomCoord).Item2, FinalList(RandomCoord).Item1)
Else
    ClickComputerMove(FinalList(RandomCoord).Item1, FinalList(RandomCoord).Item2)
End If
'Calls sub that actually clicks the button
d Sub
```

This is the code for the difficulty 3 level

The code first switches player to the human and then runs a Dijkstra's for the humans go (Runs a Dijkstra for the opposite player and nothing else). This will find the best moves for the human player which will put them closer to winning. After this it changes player back to the computer and runs a Dijkstra's again for this player. However as my Dijkstra code only goes up to down the Dijkstra for the blue player will need to be rotated 90 degrees anticlockwise. Flipboard takes care of this as it rotates the coordinates of the board 90 degrees anticlockwise

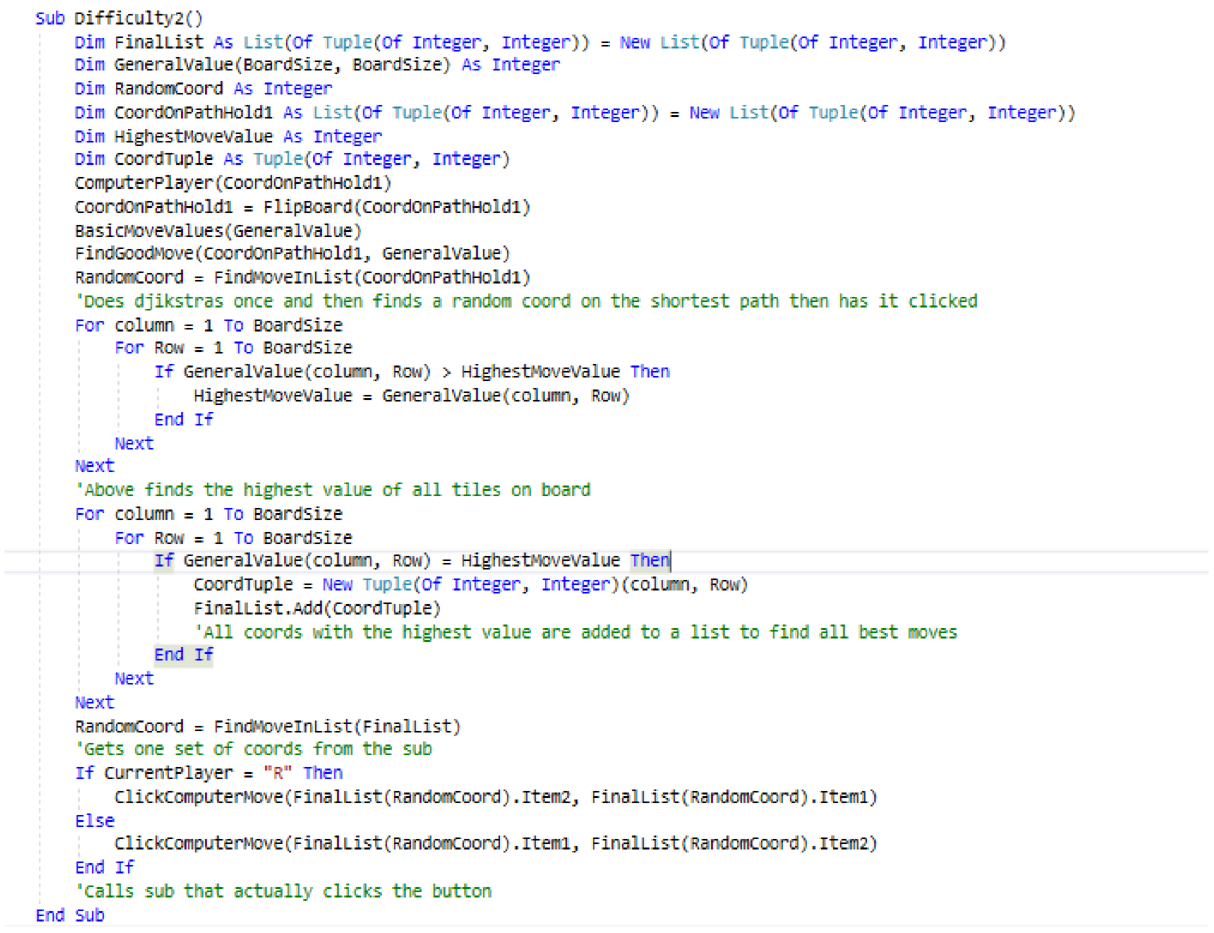
Next we find out how many times each coordinate is in the list of coordinates for both Dijkstra's appears. We then assign how many times each coordinate appears in the Dijkstra's to an array. After wards we combine this array with all the coordinates that can make bridges from the tiles of both players. This is because the bridges are the strongest strategy option you can make. This way as you look at both players Dijkstra's and bridges you will be able to find the tile which appears most in both your and the opponents shortest path and if those points will be able to make any bridges

After we have given every tile a value we find which tile/ tiles are the highest value (aka the best move). As there could be multiple tiles which have the same highest value then we get a random one of these tiles and get the computer to click it. We have made sure that the tile is free in the sub where you find the bridges that can be made. This stops it from clicking on a tile that has already been taken by giving them a negative value which will mean that they will never be chosen

As the final coordinates are set up that the blue player was going up and down. This will require you to need to flip the coordinates around if the red player is the computer. This is why there is a final if statement where if the current player is red then the first value of the coordinate is actually the second value of that coordinates and vice versa

This then calls a sub which will use those coordinates to click the button that the computer has found to be the best move

Bridges are explained on page 5



```
Sub Difficulty2()
    Dim FinalList As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
    Dim GeneralValue(BoardSize, BoardSize) As Integer
    Dim RandomCoord As Integer
    Dim CoordOnPathHold1 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
    Dim HighestMoveValue As Integer
    Dim CoordTuple As Tuple(Of Integer, Integer)
    ComputerPlayer(CoordOnPathHold1)
    CoordOnPathHold1 = FlipBoard(CoordOnPathHold1)
    BasicMoveValues(GeneralValue)
    FindGoodMove(CoordOnPathHold1, GeneralValue)
    RandomCoord = FindMoveInList(CoordOnPathHold1)
    'Does djikstras once and then finds a random coord on the shortest path then has it clicked
    For column = 1 To BoardSize
        For Row = 1 To BoardSize
            If GeneralValue(column, Row) > HighestMoveValue Then
                HighestMoveValue = GeneralValue(column, Row)
            End If
        Next
    Next
    'Above finds the highest value of all tiles on board
    For column = 1 To BoardSize
        For Row = 1 To BoardSize
            If GeneralValue(column, Row) = HighestMoveValue Then
                CoordTuple = New Tuple(Of Integer, Integer)(column, Row)
                FinalList.Add(CoordTuple)
                'All coords with the highest value are added to a list to find all best moves
            End If
        Next
    Next
    RandomCoord = FindMoveInList(FinalList)
    'Gets one set of coords from the sub
    If CurrentPlayer = "R" Then
        ClickComputerMove(FinalList(RandomCoord).Item2, FinalList(RandomCoord).Item1)
    Else
        ClickComputerMove(FinalList(RandomCoord).Item1, FinalList(RandomCoord).Item2)
    End If
    'Calls sub that actually clicks the button
End Sub
```

This is the code used for the difficulty level 2.

In this code it calls the Dijkstra sub straight away. Next it flips the board as in Dijkstra's it will go create a path from top to bottom. However as my Dijkstra code only goes up to down the Dijkstra for the blue player will need to be rotated 90 degrees anticlockwise. Flipboard takes care of this as it rotates the coordinates of the board 90 degrees anticlockwise

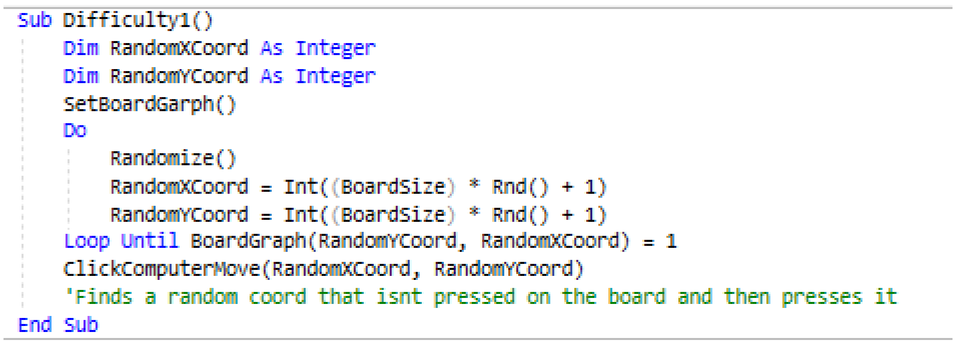
Next we find out how many times each coordinate is in the list of coordinates that make up the shortest paths. We then assign how many times each coordinate appears in that to an array. After wards we combine this array with all the coordinates that can make bridges from the tiles of both players. This is because the bridges are the strongest strategy option you can make. This way as you look at only your Dijkstra but checks both yours and your opponents bridges

After we have given every tile a value we find which tile/ tiles are the highest value. As there could be multiple tiles which have the same highest value then we get a random one of these tiles and get the computer to click it.

As the final coordinates are set up that the blue player was going up and down. This will require you to need to flip the coordinates around if the red player is the computer. This is why there is a final if statement where if the current player is red then the first value of the coordinate is actually the second value of that coordinates and vice versa

This then calls a sub which will use those coordinates to click the button that the computer has found to be the best move

Bridges are explained on page 5

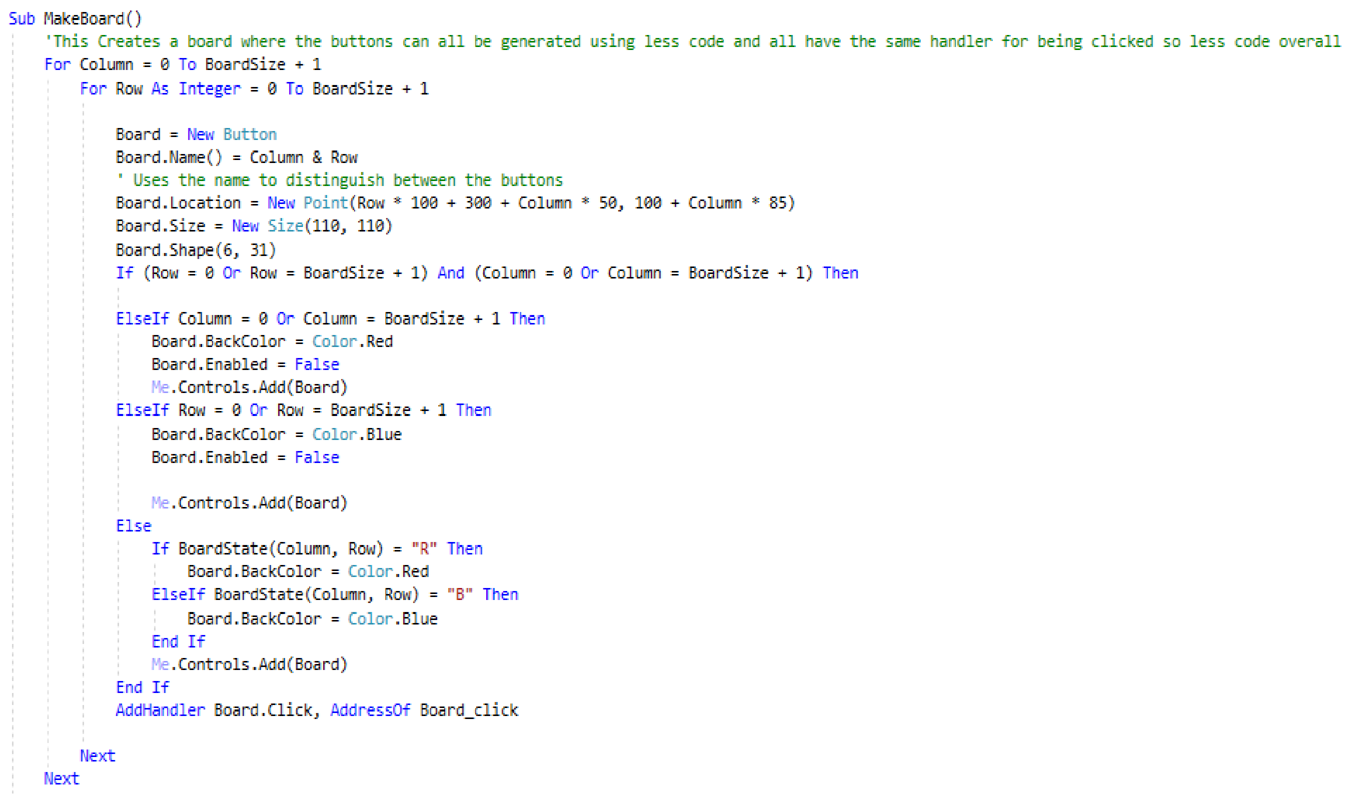


```
Sub Difficulty1()
    Dim RandomXCoord As Integer
    Dim RandomYCoord As Integer
    SetBoardGarph()
    Do
        Randomize()
        RandomXCoord = Int((BoardSize) * Rnd() + 1)
        RandomYCoord = Int((BoardSize) * Rnd() + 1)
    Loop Until BoardGraph(RandomYCoord, RandomXCoord) = 1
    ClickComputerMove(RandomXCoord, RandomYCoord)
    'Finds a random coord that isnt pressed on the board and then presses it
End Sub
```

This is the sub for the difficulty level 1

This difficulty is comprised of finding random coordinates and pressing the corresponding button

To do this I find two random numbers one for the y coordinate and one for the x coordinate. Next I loop through creating the random numbers till they correspond to a button which isn't already been pressed.



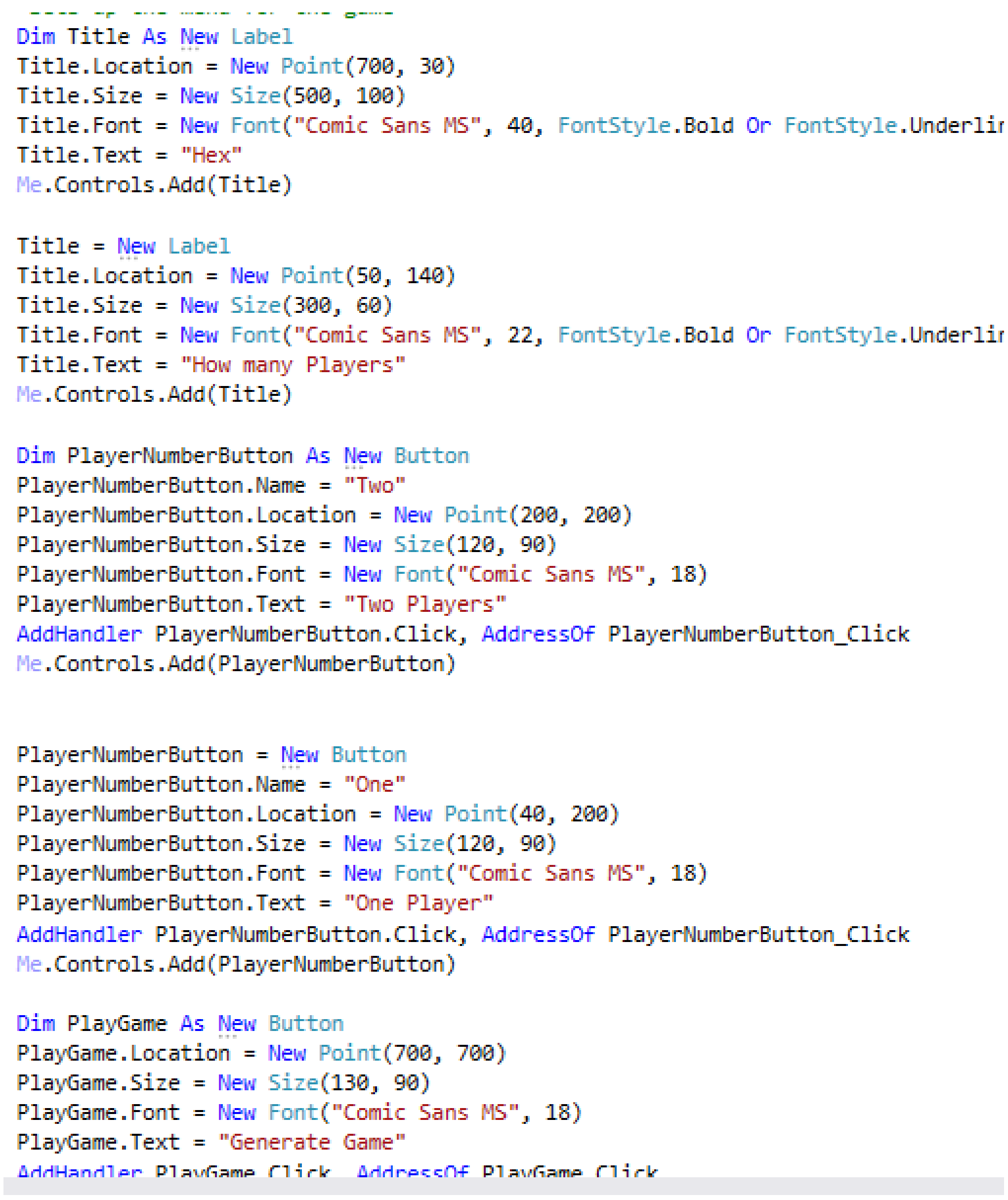
```
Sub MakeBoard()
    'This Creates a board where the buttons can all be generated using less code and all have the same handler for being clicked so less code overall
    For Column = 0 To BoardSize + 1
        For Row As Integer = 0 To BoardSize + 1

            Board = New Button
            Board.Name() = Column & Row
            ' Uses the name to distinguish between the buttons
            Board.Location = New Point(Row * 100 + 300 + Column * 50, 100 + Column * 85)
            Board.Size = New Size(110, 110)
            Board.Shape(6, 31)
            If (Row = 0 Or Row = BoardSize + 1) And (Column = 0 Or Column = BoardSize + 1) Then

            ElseIf Column = 0 Or Column = BoardSize + 1 Then
                Board.BackColor = Color.Red
                Board.Enabled = False
                Me.Controls.Add(Board)
            ElseIf Row = 0 Or Row = BoardSize + 1 Then
                Board.BackColor = Color.Blue
                Board.Enabled = False

                Me.Controls.Add(Board)
            Else
                If BoardState(Column, Row) = "R" Then
                    Board.BackColor = Color.Red
                ElseIf BoardState(Column, Row) = "B" Then
                    Board.BackColor = Color.Blue
                End If
                Me.Controls.Add(Board)
            End If
            AddHandler Board.Click, AddressOf Board_click

        Next
    Next
```

This is the code for creating the actual buttons of the board it loops through for the board size and increments the buttons so that they appear at regular intervals

You can see that I have a handler for when the buttons are clicked. This only creates one handler however all the buttons created from it will be able to use this handler. This means that there will be less repeated code. This code also has buttons which aren't able to be clicked when run these are for the edges of the board so that you know which direction the colours need to go.

There is also a check to see that if the game is being loaded from a saved game. So that if the board is being gotten from a file it will have the same colour when created as it did when the game was first saved.

Each button is given a unique name when it is created the name consists of the column number followed by the row number. The names are unique so that you can distinguish between the buttons when pressed. So when the computer has to click a button that it can find the right button to press just by having a column number and a row number.



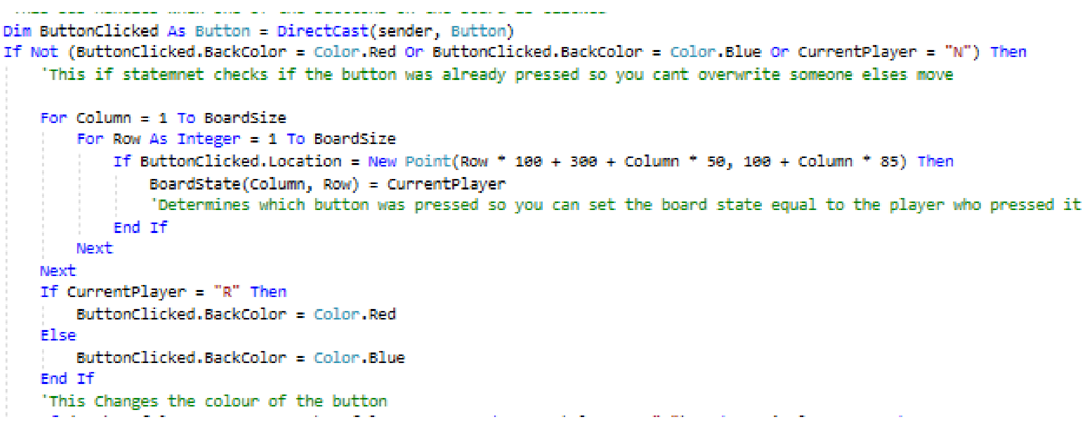
```
Dim Title As New Label
Title.Location = New Point(700, 30)
Title.Size = New Size(500, 100)
Title.Font = New Font("Comic Sans MS", 40, FontStyle.Bold Or FontStyle.Underlir
Title.Text = "Hex"
Me.Controls.Add(Title)

Title = New Label
Title.Location = New Point(50, 140)
Title.Size = New Size(300, 60)
Title.Font = New Font("Comic Sans MS", 22, FontStyle.Bold Or FontStyle.Underlir
Title.Text = "How many Players"
Me.Controls.Add(Title)

Dim PlayerNumberButton As New Button
PlayerNumberButton.Name = "Two"
PlayerNumberButton.Location = New Point(200, 200)
PlayerNumberButton.Size = New Size(120, 90)
PlayerNumberButton.Font = New Font("Comic Sans MS", 18)
PlayerNumberButton.Text = "Two Players"
AddHandler PlayerNumberButton.Click, AddressOf PlayerNumberButton_Click
Me.Controls.Add(PlayerNumberButton)


PlayerNumberButton = New Button
PlayerNumberButton.Name = "One"
PlayerNumberButton.Location = New Point(40, 200)
PlayerNumberButton.Size = New Size(120, 90)
PlayerNumberButton.Font = New Font("Comic Sans MS", 18)
PlayerNumberButton.Text = "One Player"
AddHandler PlayerNumberButton.Click, AddressOf PlayerNumberButton_Click
Me.Controls.Add(PlayerNumberButton)

Dim PlayGame As New Button
PlayGame.Location = New Point(700, 700)
PlayGame.Size = New Size(130, 90)
PlayGame.Font = New Font("Comic Sans MS", 18)
PlayGame.Text = "Generate Game"
AddHandler PlayGame.Click, AddressOf PlayGame_Click
```

In my code as well I decided to hardcode the title screen to appear when the game first loads up. This way it is much easier to transfer to other people as they don't need to set up the home screen themselves using the same Variable names I use

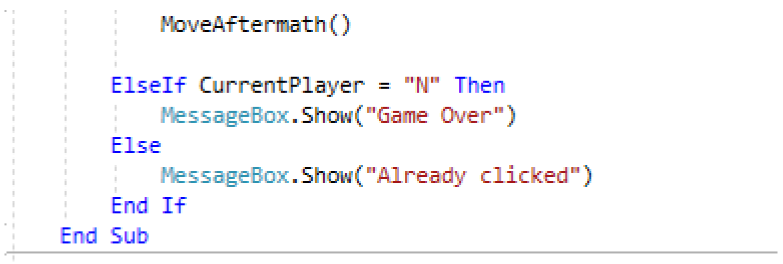
I also group my creating of the parts as it makes it much easier for other people to read and more fluent. It also will make it easier to change any of the values of the options as you can just look for the variable name and choose from there

I reuse a lot of the variables and group most of the button handlers together as it will reduce the amount of code that is needed and allows people to look at the button groups together so that they can get a better understanding of what each button does

(These are only a few of the buttons of the start screen)



```
Dim ButtonClicked As Button = DirectCast(sender, Button)
If Not (ButtonClicked.BackColor = Color.Red Or ButtonClicked.BackColor = Color.Blue Or CurrentPlayer = "N") Then
    'This if statemnet checks if the button was already pressed so you cant overwrite someone elses move

    For Column = 1 To BoardSize
        For Row As Integer = 1 To BoardSize
            If ButtonClicked.Location = New Point(Row * 100 + 300 + Column * 50, 100 + Column * 85) Then
                BoardState(Column, Row) = CurrentPlayer
                'Determines which button was pressed so you can set the board state equal to the player who pressed it
            End If
        Next
    Next
    If CurrentPlayer = "R" Then
        ButtonClicked.BackColor = Color.Red
    Else
        ButtonClicked.BackColor = Color.Blue
    End If
    'This Changes the colour of the button
```



```
            MoveAftermath()

        ElseIf CurrentPlayer = "N" Then
            MessageBox.Show("Game Over")
        Else
            MessageBox.Show("Already clicked")
        End If
    End Sub
```

This is the method used when a button is clicked. It checks at the beginning if the button is already pressed. It also calls the aftermath of a move

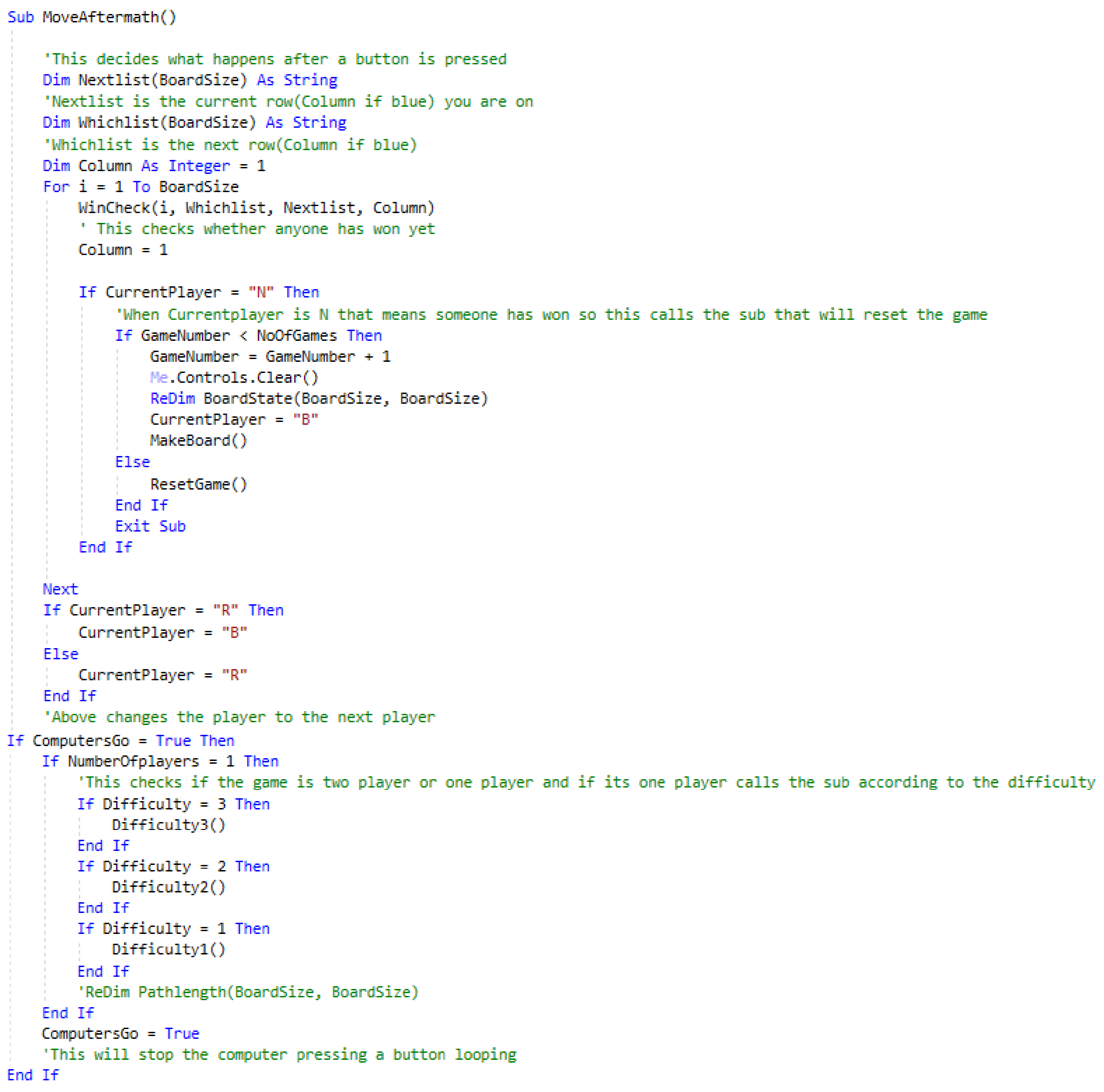
This also comes up with a message to affirm to the players that the button has already been pressed in case they didn't realise that their move wasn't valid.

I also made it so that the switching of the players was in an if statement so that if you clicked a button already that had already been chosen then you could still make a move.

I use a little check here as well to assign the correct value to the correct part of the array. I use a combination of an if statement and a for loop. It uses the same style of creating the buttons but checks if the button was pressed was equivalent to the point it's being compared to.

This use the check of seeing if the button is already coloured to see if the button has been pressed before as if the button isn't red or blue then the button won't be pressed. I also made sure to not assign a value to the lists if this check wasn't met as if it just didn't visually show but counted the move as valid then the game wouldn't be working properly.

It also assigns the colour of the button pressed in the loop so that the button will only change colour if it hasn't been pressed already.



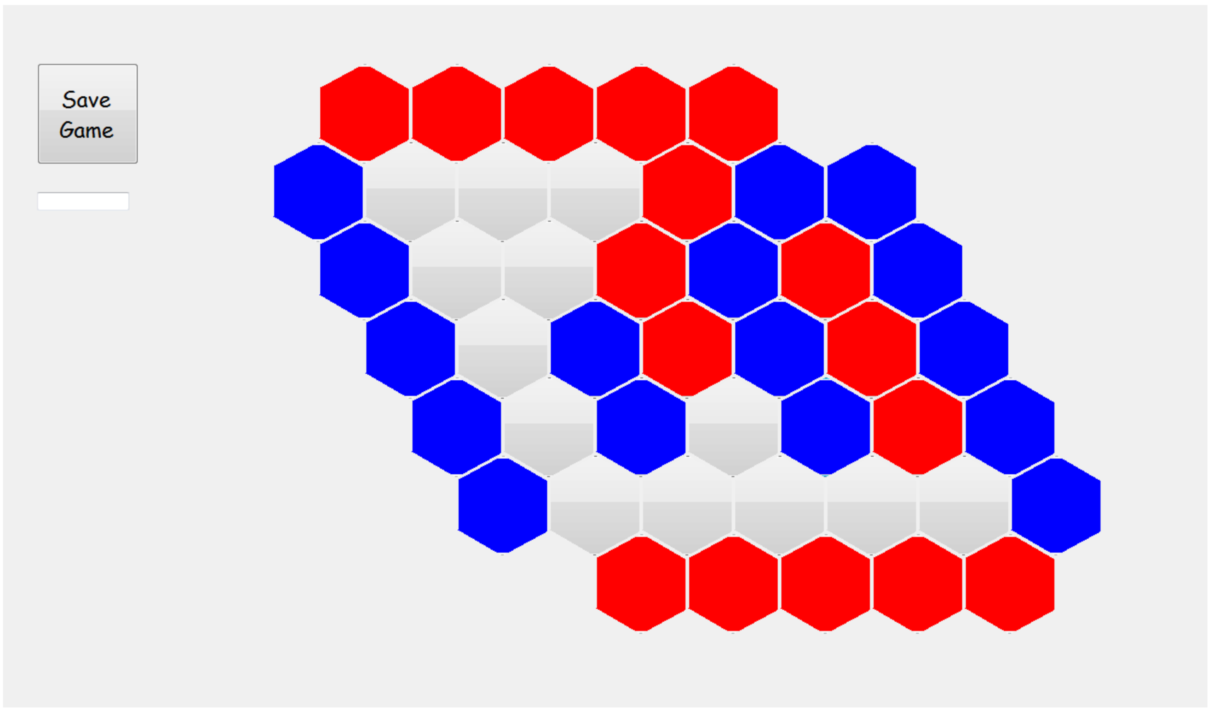
```
Sub MoveAftermath()

    'This decides what happens after a button is pressed
    Dim Nextlist(BoardSize) As String
    'Nextlist is the current row(Column if blue) you are on
    Dim Whichlist(BoardSize) As String
    'Whichlist is the next row(Column if blue)
    Dim Column As Integer = 1
    For i = 1 To BoardSize
        WinCheck(i, Whichlist, Nextlist, Column)
        ' This checks whether anyone has won yet
        Column = 1

        If CurrentPlayer = "N" Then
            'When Currentplayer is N that means someone has won so this calls the sub that will reset the game
            If GameNumber < NoOfGames Then
                GameNumber = GameNumber + 1
                Me.Controls.Clear()
                ReDim BoardState(BoardSize, BoardSize)
                CurrentPlayer = "B"
                MakeBoard()
            Else
                ResetGame()
            End If
            Exit Sub
        End If

    Next
    If CurrentPlayer = "R" Then
        CurrentPlayer = "B"
    Else
        CurrentPlayer = "R"
    End If
    'Above changes the player to the next player
If ComputersGo = True Then
    If NumberOfplayers = 1 Then
        'This checks if the game is two player or one player and if its one player calls the sub according to the difficulty
        If Difficulty = 3 Then
            Difficulty3()
        End If
        If Difficulty = 2 Then
            Difficulty2()
        End If
        If Difficulty = 1 Then
            Difficulty1()
        End If
        'ReDim Pathlength(BoardSize, BoardSize)
    End If
    ComputersGo = True
    'This will stop the computer pressing a button looping
End If
```

This is the sub for the aftermath of each move.

The first thing it does is check if the move that was made has won the game and if it has then it will either reset the game if there are still more games in the series that need to be played but if there are no more games that need to be played then the player will be asked if they want to play again or exit the game.

After the check to see if anyone has won then the player is switched so that the next player can make a move. Next it checks if it is the computers turn to move if it is then it goes to the sub of the difficulty that the player chose.

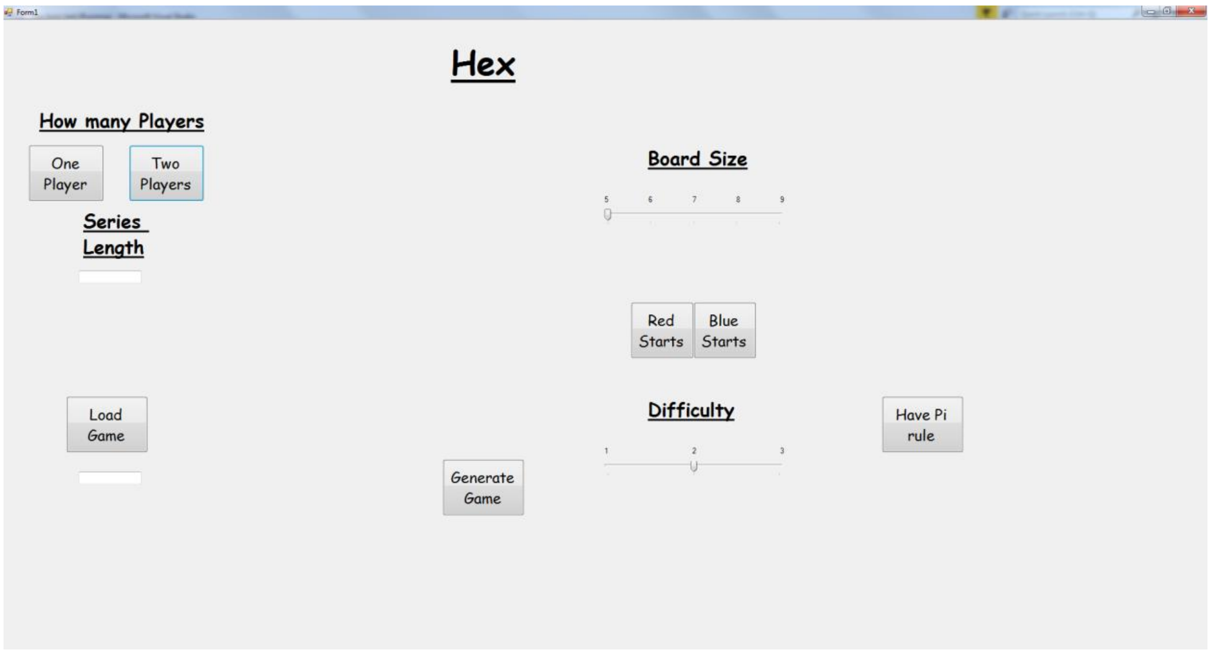
This sub is called when either a computer presses a button or when the human player clicks a button this means that there will be less repeated code and this works as the aftermaths for a human and computer should be the same as they both want to check if someone has won or not first as everything after that becomes irrelevant if there is a winner



The picture above shows what my board looks like for a board size of 5. I decided to use the colours the game typically used of blue and red and kept the buttons that aren't pressed grey. This means it is very obvious which buttons have been pressed and which haven't as the colours are very different from one another. The grey from the background and the buttons is also different which makes is easy to tell the limits of the board. The buttons also have a little space in-between them as this can make it easier to distinguish which button you are actually pressing so there will be less chance of you pressing the wrong one.

All the buttons are the same size and shape and the lines parallel to each other this in my opinion makes the board nicer to look at and easier to understand. The choose game type and Board size wouldn't be there in the actual finished display of the board and would instead have the image below to the side of it. When the game has been won there will then be a message box that appears and says who the winner of the game was. The Edges of the board will also be changed so that it is very easy to tell which direction the different players want to go in.

GUI when game is first loaded



When the game is first loaded there will be the title of the game in the middle at the top in big letters. The grey boxes represent buttons and the yellow boxes represent boxes where the user enters a numerical value.

All the buttons and text will be coded in to show up when the form is loaded. This way the player can see all the different options that they can change between for unique games.

I decided to make Ai difficulty and board size where the user changes a slider as if they were represented by buttons it would require a large amount of buttons as there is a range of board sizes that could be chosen and a range of difficulties.

The buttons change colour once pressed so that you can see which choices you have clicked on and if you actually have clicked on them.

The GUI will just be composed of labels buttons, sliders and input boxes so should be quite simple and easy for people to understand. They will also have the labels for what they are on them or above them so you can easily tell what goes where.

Validation

I use some validation when the buttons are pressed to see if they have been pressed before this makes it so you can't go where someone else has gone before which is important as you can't do that in the actual game as well

I also have validation for when the user enters board size and ai difficulty as there will be only certain values that work. This is because too large of a board will not be able to fit properly onto a computer screen and a small board wouldn't really be a game of hex. The ai difficulty also will be from 1-3 So there can't be any values inputted that are below 1

There is also validation that if no values are entered into game then a game will still run but with default values so that way you don't have to change every single option once to get a working game

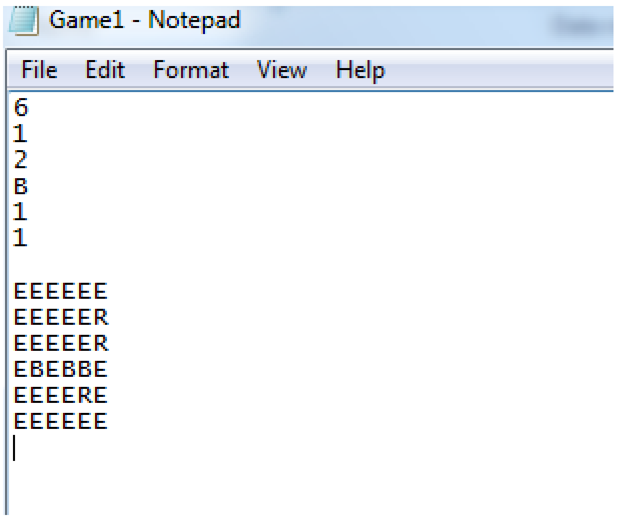
Storage

There is also a save game and load game feature. This allows the player to come back and try different strategies. It allows the person to enter a name that they want to save the game as and a name that they want to load. This means that the person can assign whatever names to games that have interested them.

There is a try statement for the load up of the game which allows the program not to crash if it loads up a file that doesn't exist

The save game features stores the size of the board of the game, how many players the game consisted of, Difficulty of the computer, player it currently is, which game you are on, how many games are in the series and what the board looks like.

When a game is loaded the board state is changed to look what liked when it was saved



```
6
1
2
B
1
1

EEEEEE
EEEEER
EEEEER
EBEBBE
EEEERE
EEEEEE
```

The above picture is what a saved game is stored as

The first row contains the size of the board of the game

The second row contains how many players the game consisted of

The third row Shows the Difficulty of the computer

The fourth row Contains which player it currently is

Fourth row conatins which game you are on

Fifth row contains how many games are in the series

The last block of letters is what the board looks like the E represents a button that hasn't been pressed yet so doesn't have a colour.

Testing strategy

Input for Board size

https://www.youtube.com/watch?v=9lEJW-1tg6c

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	7	Test how to make the size of the board	Typical	Creates a 7 by 7 board	Pass
2	6		Typical	Creates a 6 by 6 board	Pass
3			Erroneous	Creates a 5 by 5 board	Pass

4	5		Extreme	Creates a 5 by 5 board	Pass
5	9		Extreme	Creates a 8 by 8 board	Pass

Input for AI difficulty

https://www.youtube.com/watch?v=9lEJW-1tg6c

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	2	Test that the path finding algorithm with look for the shortest path for the current player	Typical	Performs 1 Dijkstra algorithm	Pass
4			Typical	Performs 1 Dijkstra algorithm	Pass
5	3		Extreme	Performs 2 Dijkstra algorithm on the board	Pass
7	1		Extreme	Generates random coordinates to make a move	Pass

Input for series length

https://www.youtube.com/watch?v=9lEJW-1tg6c

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	1	Test that the amount of games in the series will be equal to what the user enters or if no	Typical	1 game then asks about playing again or rule changes	Pass

		valid entry 1 games			
2	3		Typical	3 game then asks about playing again or rule changes	Pass
3			Erroneous	1 game then asks about playing again or rule changes	Pass
4	5		Extreme	5 game then asks about playing again or rule changes	Pass

Check to see if pi rule works

https://www.youtube.com/watch?v=9lEJW-1tg6c

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	Clicked button to give pi rule	Test that the pi rule will be activated for the game	Typical	Button changes colour and gives you a choice to switch or stay after the first move	Pass
2	Didn't click		Typical	Doesn't give you the option to switch or stay	Pass
3	Clicks button for Pi rule twice			Button changes colour and gives you a choice to switch or stay after the first move	Pass

Check to see if Save game works

https://www.youtube.com/watch?v=sYUPDtZewe0

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	Typed Hex1 and clicked save game	Test that a file will be saved with the name of the thing that was typed in	Typical	File appears with the necessary details that need to be loaded to recreate the game. File has the name Hex1	Pass
2	Typed 2 and clicked save game		Typical	File appears with the necessary details that need to be loaded to recreate the game. File has the name 2	Pass
3	Typed Hex1 and clicked save game when already a Hex 1 saved			File appears with the necessary details that need to be loaded to recreate the game. File has the name Hex1 and has overwritten the old hex 1	Pass
4	Saved a game with no text where you enter the name you are going to save it as			File appears with the necessary details that need to be loaded to recreate the game. File has no name	Pass

Check to see if load Game works.

https://www.youtube.com/watch?v=sYUPDtZewe0

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	Typed Hex1 and clicked load game	Test that a file will be loaded with the name of the thing that was typed in. And has the game from when it was clicked save	Typical	Loads the game Hex1 that was saved already. Current player was what it was when the game was loaded, board was in the same state and difficulty and numbers of players are the same. Number of games in the series is also the same	Pass
2	Typed 2 and clicked load game		Typical	Loads the game 2 that was saved already. Current player was what it was when the game was loaded, board was in the same state and difficulty and numbers of players are the same. Number of games in the series is also the same	Pass

3	Typed Hex2 and clicked save game when there isn't a file called Hex 2 saved			Nothing happens, No game is loaded and no message appears	Pass
4	Typed and clicked load game			Loads the game that has a file name of no characters that was saved already. Current player was what it was when the game was loaded, board was in the same state and difficulty and numbers of players are the same. Number of games in the series is also the same	Pass

Check if can change starting player

https://www.youtube.com/watch?v=9lEJW-1tg6c

Test Number	Input	Description	Data type	Expected Result	Pass Fail

1	Clicked on blue starts	Test that the starting player will be the player that button was pressed	Typical	Blue player starts	Pass
2	Clicked on red starts when it is one player		Typical	Red button changes colour and computer goes first and is red	Pass
3	Clicked on red goes first when it is two players			Red button changes colour and red player will start the game.	Pass

Check To see if gamewon sub works

https://www.youtube.com/watch?v=K4R-DEmxfMc

Test Number	Input	Description	Data type	Expected Result	Pass Fail
1	Straight line down with a winning move	These tests are designed to show that all paths will return a win if a player has just made a winning move	Typical	Returns the colour of the player who has won	Pass
2	Straight line with a different board size	This test is to show that it works for all different board sizes and will return if someone has won	Typical	Returns the colour of the player who has won	Pass
3	Have a winning path that goes right and is a complete path			Returns the colour of the player who has won	Pass
4	Have a winning path that that goes left at one point			Returns the colour of the player who has won	Pass
5	Have a winning path that that goes left at one point and then right a later point			Returns the colour of the player who has won	Pass
6	Have a winning path that that goes left at one point and then left again at a later point			Returns the colour of the player who has won	Pass

	on the path				
7	Have a winning path that that goes right at one point and then right a later point			Returns the colour of the player who has won	Pass
8	Have a winning path that that goes right at one point and then left again at a later point on the path			Returns the colour of the player who has won	Pass

Check to see that clicking yes after playing a game then instantly clicking generate game will give you the same rules that you just played with-

https://www.youtube.com/watch?v=9lEJW-1tg6c

Step through of Dijkstra’s to see that it works

https://www.youtube.com/watch?v=YR2eVCtuzKE&t=1s

https://www.youtube.com/watch?v=CsyVf3an4PI&t=436s

Full step through of won game check to show it works

https://www.youtube.com/watch?v=pjW7bOD0EkI

https://www.youtube.com/watch?v=XBjo4G5uJWM

Requirements

Requirements	Met	Proof
1. Game is launched 1.1. Options are displayed 1.1.1.Board size 1.1.2.AI difficulty 1.1.3.Number of games 1.1.4.Who goes first 1.1.5.Use Pi rule 1.1.6.Load Game	All the requirements in step one have been fulfilled as all the options that are wanted are displayed.	Proof is in design showing game when loaded up
1.1. User can press options 1.2. Buttons change colour when they are chosen	All the buttons which you can choose between do change colour to give the person that pressed them an idea of what has been pressed.	Proof is in test -Check to see if pi rule works and red player goes first
2. Hex board is created 2.1. Board is the shape that the user picked 2.2. Board is the size the user picked	Board is also generated correctly each time a game is loaded which means that the requirement to have a board that the user wanted has been fulfilled. And the board also has visual clues on which way the colours go so that it is easier on load up to see how to play your colour.	Proof in testing board size https://www.youtube.com/watch?v=9lEJW-1tg6c
3. Player chosen to go first goes 3.1. Button changes colour to their colour 3.1.1.The button pressed changes colour to the current player	The game also plays correctly as the buttons on the board will change colour depending on the player whose go it is. You can also not over click someone else's button once it's been clicked. This means that the game plays like it would in person. The computer can also play against a person with three different levels of difficulty. This will meet the requirement of having it so that you can play the game	https://www.youtube.com/watch?v=9lEJW-1tg6c

	by yourself. The game also works as it would in real life if there is only one player as you still can't override other people's moves and the computer can't override moves as well.	
4. Second player is asked about pie rule 4.1. They let the play stand 4.2. They switch goes with the original player	Game presents you with two buttons after the first move if the pie rule is active. The buttons state stay or switch. If you click the stay button then the game continues onwards from there. However if switch is pressed then the second player now takes the role of the first player and gets there move while the first player will then have to go again	https://www.youtube.com/watch?v=9lEJW-1tg6c
5. Second player now goes	Second player will be able to make a move after the first player and will have whatever button they press turn to the associated colour with them	
6. Game goes until someone wins	The game will also automatically stop when a player has won. This means that you don't have to say yourself if there is a winner.	https://www.youtube.com/watch?v=pjW7bOD0EkI
7. Has different difficulties 7.1. Difficulty one is the easiest difficulty to beat- Moves chosen at random 7.2. Difficulty two uses Dijkstra's for the current player to find the shortest path 7.2.1.Runs Dijkstra for yourself 7.2.2.Chooses a move which is on the shortest path 7.3. Difficulty three uses a Dijkstra for your opponent and yourself	The game has three difficulties which you can choose between on a slider. The first difficulty consists of the tile that the computer choosing is random. The tile is always free and hasn't been pressed yet The second difficulty runs a Dijkstra algorithm on the board and finds the path lengths of all the tiles on the board. Next a random tile which lies on any of the shortest paths is chosen at random that is free	https://www.youtube.com/watch?v=YR2eVCtuzKE&t=1s

7.3.1.Runs Dijkstra for yourself 7.3.2.Runs Dijkstra for opponent 7.3.3.Also considers which pieces are around it in order to prioritize making bridges 7.3.4.Combines all three to form a move which best blocks your opponent, creates a bridge and helps complete your path 7.3.5.Chooses the move that does all the things above	The third difficulty consists of running a Dijkstra for you and your opponent. These two Dijkstra's are then combined to find the shortest path lengths between them (tiles which appear in both shortest paths). These values are then combined with all the places that bridges (for what bridges are look at research and strategy) can be made. After all these vlasue are calculated they are then added together to find which tile can create the most bridges and lie on the shortest paths.	
8. Player Chooses save game 8.1. Allows User to choose a Name to save the game as 8.1.1.Saves the current player 8.1.2.Saves The board size, difficulty of computer, Number of players, tiles already chosen	There is also a save game feature in my code which allows you to save multiple games at any state. This will allow the player to come back and try multiple strategies to see how different ways of playing compare against each other.. You can also create your own names for the files this means that you set up your own system that you remember them by.	https://www.youtube.com/watch?v=sYUPDtZewe0
9. Load Game 9.1. Game is loaded with all the relevant information 9.2. Uses the name of the file to load the game	You can also load the games up and the features of those games are saved so that everything is like how it would be if you didn't stop playing. The way the games are loaded is by typing the name of the game and hitting load game. This will then create a board which is identical to when the game was saved	https://www.youtube.com/watch?v=sYUPDtZewe0

10. Game checks if a player has won 10.1. Looks at button that was just pressed 10.1.1. If it has any of the same colour next to it checks the buttons around that one 10.1.2. Goes through the lists to see if there is a straight path through 10.2. If the game hasn't been won then players keep making moves until it has 10.3. When game has been won displays who won	Uses recursion to check all the available paths from a piece to see if there is a complete path from one side to another. It first checks downwards then checks to the left and finally to the right of the tile it is on. If there has been a winner a sub is called which displays the colour of the person who won.	https://www.youtube.com/watch?v=pjW7bOD0EkI
11. If there are more games in the series 11.1. Repeat steps from 2 to 7 11.2. Once the series has been won display the score and winner	The series element also works mostly. The series will consist of the right number of games that the person has typed in and each of those games will have the same board size, difficulty of computer, number of players. However only the first game will have the pi rule if the pi rule is chosen and the first game and only the first game will have red go first if red is chosen to go first.	https://www.youtube.com/watch?v=9lEJW-1tg6c
12. Ask the player if they want to play again 12.1. If they say no exit the game 12.2. If they say yes give options 12.2.1. They can either play with same settings 12.2.2. Or take them back to step 1	Once a series is done the user is presented with two options play again or don't play again. If they play again and instantly press generate game then they will have the same settings so the requirement to play with the same settings is met.	https://www.youtube.com/watch?v=9lEJW-1tg6c

Improvements

Improvements- If I revisited this problem I would change up the difficulties and add a fourth difficulty which adds one extra option to consider choosing a best move this option would consist of constant values for the place on the board. For example having the middle tile be the highest value and decreasing value from the middle of the board. This would help the ai at the beginning of the game to have a stronger start as they would start from the middle which can give a better option for paths to go along.

Other options in the menu I would add if I revisited this is to add a way to change the boards shape and to have different colours. However the colours won't change gameplay at all but can just be added to make the game more intriguing. However having a way to change board shape would need a new way to check if someone has won due to the fact that it won't be across or down to win the game but some other direction.

Interview for final piece

I am interviewing Colin Zack who is the client for the game about how he finds the final product of the game of hex I made.

How do you feel about the different levels of difficulty?

I am happy with the different levels of difficulty that come with the game. They can play well and are quite hard to beat if they are on the highest level if you know the strategies to play the game. The lowest level I feel is quite good for beginners as it allows them to experiment with different strategies and the second difficulty is nice to help develop those strategies. I also like the use of a slider to have the choice for difficulty as it is very easy to change between them

How do you like the interface for the game and options?

I like the interface as it is quite simple and minimalistic which makes it very easy to find all the different settings to choose from at the beginning of the game. I particularly like how the buttons change colour when they are pressed this makes it very nice to see what options are chosen. There are lots of different options as well which makes the replay ability of the game quite high. Once the game is actually loaded up the board looks quite nice and having it so that the colours of the sides are there makes it easy to see what direction the colours need to go which makes it easy to find out when you only just start using this.

Is the Pi rule added as you would like?

Pi rule was implemented very well as you get the options to switch or stay and even the computer player has a choice of these. This can add a new level of complexity which allows you to think even more about the start of the game and how to play around having the ability to have your move taken from you. I also like how the pi rule isn't on by default as it isn't typically a naturally a rule in most common games of hex that I play this way I don't need to turn it off when I start a game all the time

How do you feel about no custom board shapes?

I'm not too disappointed about not having custom board shapes as they change up the game at a very structural level so it still has lots of playability and different ways of playing with just regular hex instead of having different shapes.

Do you like the Save and load feature of the game?

Yes I am very happy that these features are in the game as it allows me to play over games multiple times if I want to. I also like the way that the names are saved to what you want to call them which means that I can group games into different ways depending on what's happened in them so that I can look at multiple games which are similar. The fact that the games are loaded up as they were left up and you can load up the same state multiple times is very nice as it means you can redo it multiple times

Code

```
Public Class Form1
    Dim BoardSize As Integer = 5
    Private CurrentPlayer As Char = "B"
    Dim Difficulty As Integer = 2
    Dim NoOfGames As Integer = 1
    Dim GameNumber As Integer = 1
    Dim WinnerOfGames(NoOfGames - 1)

    Dim BoardState(BoardSize, BoardSize)
    Dim BoardGraph(BoardSize, BoardSize)
    Dim NumberOfplayers As Integer = 1

    Dim ComputersGo As Boolean = True
    Dim Board As New Button

    Dim UsePiRule As Boolean = False



    Dim SavedFileName As String





    Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.WindowState = FormWindowState.Maximized
        CreateMenu()
    End Sub
    Sub CreateMenu()
        'Sets up the menu for the game
```

```
Dim Title As New Label
Title.Location = New Point(700, 30)
Title.Size = New Size(500, 100)
Title.Font = New Font("Comic Sans MS", 40, FontStyle.Bold Or FontStyle.Underline)
Title.Text = "Hex"
Me.Controls.Add(Title)

Title = New Label
Title.Location = New Point(50, 140)
Title.Size = New Size(300, 60)
Title.Font = New Font("Comic Sans MS", 22, FontStyle.Bold Or FontStyle.Underline)
Title.Text = "How many Players"
Me.Controls.Add(Title)

Dim PlayerNumberButton As New Button
PlayerNumberButton.Name = "Two"
PlayerNumberButton.Location = New Point(200, 200)
PlayerNumberButton.Size = New Size(120, 90)
PlayerNumberButton.Font = New Font("Comic Sans MS", 18)
PlayerNumberButton.Text = "Two Players"
AddHandler PlayerNumberButton.Click, AddressOf PlayerNumberButton_Click
Me.Controls.Add(PlayerNumberButton)


PlayerNumberButton = New Button
PlayerNumberButton.Name = "One"
PlayerNumberButton.Location = New Point(40, 200)
PlayerNumberButton.Size = New Size(120, 90)
PlayerNumberButton.Font = New Font("Comic Sans MS", 18)
PlayerNumberButton.Text = "One Player"
AddHandler PlayerNumberButton.Click, AddressOf PlayerNumberButton_Click
```

```
Me.Controls.Add(PlayerNumberButton)

Dim PlayGame As New Button
PlayGame.Location = New Point(700, 700)
PlayGame.Size = New Size(130, 90)
PlayGame.Font = New Font("Comic Sans MS", 18)
PlayGame.Text = "Generate Game"
AddHandler PlayGame.Click, AddressOf PlayGame_Click
Me.Controls.Add(PlayGame)

Dim LoadGame As New Button
LoadGame.Location = New Point(100, 600)
LoadGame.Size = New Size(130, 90)
LoadGame.Font = New Font("Comic Sans MS", 18)
LoadGame.Text = "Load Game"
AddHandler LoadGame.Click, AddressOf LoadGame_Click
Me.Controls.Add(LoadGame)

Dim PiRule As New Button
PiRule.Location = New Point(1400, 600)
PiRule.Size = New Size(130, 90)
PiRule.Font = New Font("Comic Sans MS", 18)
PiRule.Text = "Have Pi rule"
AddHandler PiRule.Click, AddressOf PiRule_Click
Me.Controls.Add(PiRule)

Dim StartingPlayer As New Button
StartingPlayer.Name = "Red"
StartingPlayer.Location = New Point(1000, 450)
StartingPlayer.Size = New Size(100, 90)
StartingPlayer.Font = New Font("Comic Sans MS", 18)
StartingPlayer.Text = "Red Starts"
AddHandler StartingPlayer.Click, AddressOf StartingPlayer_Click
Me.Controls.Add(StartingPlayer)

StartingPlayer = New Button
StartingPlayer.Name = "Blue"
StartingPlayer.Location = New Point(1100, 450)
StartingPlayer.Size = New Size(100, 90)
StartingPlayer.Font = New Font("Comic Sans MS", 18)
StartingPlayer.Text = "Blue Starts"
AddHandler StartingPlayer.Click, AddressOf StartingPlayer_Click
Me.Controls.Add(StartingPlayer)



Title = New Label
Title.Location = New Point(1020, 200)
Title.Size = New Size(200, 50)
Title.Font = New Font("Comic Sans MS", 22, FontStyle.Bold Or FontStyle.Underline)
Title.Text = "Board Size"
Me.Controls.Add(Title)



Dim Slider As New TrackBar
Slider.Location = New Point(950, 300)
Slider.Size = New Size(300, 60)
Slider.Minimum = 5
Slider.Maximum = 9
Slider.SmallChange = 1
Slider.LargeChange = 1
Me.Controls.Add(Slider)
AddHandler Slider.ValueChanged, AddressOf Slider_Change
```

```
Title = New Label
Title.Location = New Point(955, 280)
Title.Text = "5"
Me.Controls.Add(Title)

Title = New Label
Title.Location = New Point(1025, 280)
Title.Text = "6"
Me.Controls.Add(Title)
Title.BringToFront()

Title = New Label
Title.Location = New Point(1095, 280)
Title.Text = "7"
Me.Controls.Add(Title)
Title.BringToFront()

Title = New Label
Title.Location = New Point(1165, 280)
Title.Text = "8"
Me.Controls.Add(Title)
Title.BringToFront()

Title = New Label
Title.Location = New Point(1235, 280)
Title.Text = "9"
Me.Controls.Add(Title)
Title.BringToFront()

Slider = New TrackBar
```

```
Slider.Location = New Point(950, 700)

Slider.Size = New Size(300, 60)

Slider.Minimum = 1

Slider.Maximum = 3

Slider.Value = 2

Slider.SmallChange = 1

Slider.LargeChange = 1

Me.Controls.Add(Slider)

AddHandler Slider.ValueChanged, AddressOf ComputerLevel


Title = New Label

Title.Location = New Point(955, 680)

Title.Text = "1"

Me.Controls.Add(Title)


Title = New Label

Title.Location = New Point(1095, 680)

Title.Text = "2"

Me.Controls.Add(Title)

Title.BringToFront()


Title = New Label

Title.Location = New Point(1235, 680)

Title.Text = "3"

Me.Controls.Add(Title)

Title.BringToFront()



Title = New Label

Title.Location = New Point(1020, 600)

Title.Size = New Size(200, 50)
```

```
        Title.Font = New Font("Comic Sans MS", 22, FontStyle.Bold Or FontStyle.Underline)
        Title.Text = "Difficulty"
        Me.Controls.Add(Title)



        Dim FileName As New TextBox
        FileName.Location() = New Point(120, 720)
        AddHandler FileName.Leave, AddressOf FileName_Click
        Me.Controls.Add(FileName)

        Title = New Label
        Title.Location = New Point(120, 300)
        Title.Size = New Size(200, 100)
        Title.Font = New Font("Comic Sans MS", 22, FontStyle.Bold Or FontStyle.Underline)
        Title.Text = "Series Length"
        Me.Controls.Add(Title)

        Dim GameCount As New TextBox
        GameCount.Location() = New Point(120, 400)
        AddHandler GameCount.Leave, AddressOf GameCount_Click
        Me.Controls.Add(GameCount)



    End Sub

    Private Sub StartingPlayer_Click(sender As Object, e As EventArgs)
        Dim StartingPlayer As Button = DirectCast(sender, Button)

        If StartingPlayer.Name = "Blue" Then
            CurrentPlayer = "B"
            ComputersGo = True
            StartingPlayer.BackColor = Color.DimGray
            Board = FindFromButtons("Red")
            Board.UseVisualStyleBackColor = True
        Else
            CurrentPlayer = "R"
            StartingPlayer.BackColor = Color.DimGray
            Board = FindFromButtons("Blue")
            Board.UseVisualStyleBackColor = True
        End If
    End Sub

    Private Sub PiRule_Click(sender As Object, e As EventArgs)
        Dim PiRule As Button = DirectCast(sender, Button)
        If UsePiRule = False Then
            UsePiRule = True
            PiRule.BackColor = Color.DimGray
        Else
            UsePiRule = False
            PiRule.UseVisualStyleBackColor = True
        End If

    End Sub

    Private Sub GameCount_Click(sender As Object, e As EventArgs)
        Dim GameCount As TextBox = DirectCast(sender, TextBox)
        GameCount.Update()
        NoOfGames = GameCount.Text()
        ReDim WinnerOfGames(NoOfGames - 1)
    End Sub
    Private Sub ComputerLevel(sender As Object, e As EventArgs)
        'Sets the difficulty of the computer
        Dim Slider As TrackBar = DirectCast(sender, TrackBar)

        Difficulty = Slider.Value

    End Sub

    Private Sub LoadGame_Click()

        Dim Load As System.IO.StreamReader

        Try

            Load = My.Computer.FileSystem.OpenTextFileReader(SavedFileName & ".txt")

            BoardSize = Load.ReadLine()

            'ReDim HoldLastRow(BoardSize)

            'ReDim Paths(BoardSize, BoardSize)

            'ReDim Pathlength(BoardSize, BoardSize)

            ReDim BoardState(BoardSize, BoardSize)

            ReDim BoardGraph(BoardSize, BoardSize)

            NumberOfplayers = Load.ReadLine()

            Difficulty = Load.ReadLine()

            CurrentPlayer = Load.ReadLine()

            GameNumber = Load.ReadLine()

            NoOfGames = Load.ReadLine()

            ReDim WinnerOfGames(NoOfGames)

            For i = 0 To WinnerOfGames.Count - 1


                WinnerOfGames(i) = Chr(Load.Read())

            Next


            For Column = 1 To BoardSize

                For Row As Integer = 1 To BoardSize

                    BoardState(Column, Row) = Chr(Load.Read())


                Next

                Load.ReadLine()

            Next
            Load.Close()
            For Column = 1 To BoardSize
                For Row As Integer = 1 To BoardSize
                    If BoardState(Column, Row) = "E" Then
                        BoardState(Column, Row) = Nothing
                    End If
                Next
            Next
            Me.Controls.Clear()
            MakeBoard()
        Catch Exception As Exception

        End Try
    End Sub
    Private Sub PlayGame_Click(sender As Object, e As EventArgs)
        'Clears menu and calls the make board sub
        Me.Controls.Clear()
        MakeBoard()
    End Sub

    Private Sub Slider_Change(sender As Object, e As EventArgs)
        'Changes the size of the board according to the value of the slider and resets all things that rely
on boardsize so they fit the new boardsize
        Dim Slider As TrackBar = DirectCast(sender, TrackBar)
        BoardSize = Slider.Value
        'ReDim HoldLastRow(BoardSize)
        'ReDim Paths(BoardSize, BoardSize)
        'ReDim Pathlength(BoardSize, BoardSize)
        ReDim BoardState(BoardSize, BoardSize)
        ReDim BoardGraph(BoardSize, BoardSize)
    End Sub


    Private Sub PlayerNumberButton_Click(sender As Object, e As EventArgs)
        'Gets the button that was pressed and sees if it was one player or two player then makes the game that number of players
        Dim NoOfPlayers As Button = DirectCast(sender, Button)
        If NoOfPlayers.Name = "One" Then
            NumberOfplayers = 1
            NoOfPlayers.BackColor = Color.DimGray
            Board = FindFromButtons("Two")
            Board.UseVisualStyleBackColor = True
        Else
            NumberOfplayers = 2
            NoOfPlayers.BackColor = Color.DimGray
            Board = FindFromButtons("One")
            Board.UseVisualStyleBackColor = True
        End If
    End Sub


    Private Function FindFromButtons(ByVal Name As String)


        Dim buttons = From c In Controls Where TypeOf (c) Is Button Select c
        Dim G = (From C In buttons Where C.Name() = Name Select C).FirstOrDefault
        Board = G
        Return Board
    End Function


    Sub MakeBoard()
        'This Creates a board where the buttons can all be generated using less code and all have the same handler for being clicked so less code overall
        For Column = 0 To BoardSize + 1
            For Row As Integer = 0 To BoardSize + 1
            Board = New Button
            Board.Name() = Column & Row
            ' Uses the name to distinguish between the buttons
            Board.Location = New Point(Row * 100 + 300 + Column * 50, 100 + Column * 85)
            Board.Size = New Size(110, 110)
            Board.Shape(6, 31)
            If (Row = 0 Or Row = BoardSize + 1) And (Column = 0 Or Column = BoardSize + 1) Then

            ElseIf Column = 0 Or Column = BoardSize + 1 Then
                Board.BackColor = Color.Red
                Board.Enabled = False
                Me.Controls.Add(Board)
            ElseIf Row = 0 Or Row = BoardSize + 1 Then
                Board.BackColor = Color.Blue
                Board.Enabled = False

                Me.Controls.Add(Board)
            Else
                If BoardState(Column, Row) = "R" Then
                    Board.BackColor = Color.Red
                ElseIf BoardState(Column, Row) = "B" Then
                    Board.BackColor = Color.Blue
                End If
                Me.Controls.Add(Board)
            End If
            AddHandler Board.Click, AddressOf Board_click

        Next
    Next
    Dim SaveGame As New Button
```

```
SaveGame.Location() = New Point(100, 100)
SaveGame.Size = New Size(110, 110)
SaveGame.Text = "Save Game"
SaveGame.Font = New Font("Comic Sans MS", 18)
AddHandler SaveGame.Click, AddressOf SaveGame_Click
Me.Controls.Add(SaveGame)

Dim FileName As New TextBox
FileName.Location() = New Point(100, 240)
AddHandler FileName.Leave, AddressOf FileName_Click
Me.Controls.Add(FileName)

If CurrentPlayer = "R" Then
   CurrentPlayer = "B"
   MoveAftermath()
   If UsePiRule = True Then
      Dim Switch As New Button
      Switch.Location = New Point(800, 800)
      Switch.Size = New Size(130, 90)
      Switch.Font = New Font("Comic Sans MS", 18)
      Switch.Text = "Switch"
      Switch.Name = "Switch"
      AddHandler Switch.Click, AddressOf Switch_Click
      Me.Controls.Add(Switch)
      Dim Stay As New Button
      Stay.Location = New Point(500, 800)
      Stay.Size = New Size(130, 90)
      Stay.Font = New Font("Comic Sans MS", 18)
      Stay.Text = "Stay"
      Stay.Name = "Stay"
      AddHandler Stay.Click, AddressOf Stay_Click
```

```
            Me.Controls.Add(Stay)
            UsePiRule = False
        End If
    End If

End Sub

Private Sub Switch_Click(sender As Object, e As EventArgs)
    Dim Switch As Button = DirectCast(sender, Button)
    Board = FindFromButtons("Stay")
    Me.Controls.Remove(Board)
    Me.Controls.Remove(Switch)
    If CurrentPlayer = "B" Then
        CurrentPlayer = "R"
    Else
        CurrentPlayer = "B"
    End If
    MoveAftermath()
End Sub
Private Sub Stay_Click(sender As Object, e As EventArgs)
    Dim Stay As Button = DirectCast(sender, Button)
    Board = FindFromButtons("Switch")
    Me.Controls.Remove(Board)
    Me.Controls.Remove(Stay)
End Sub

Private Sub FileName_Click(sender As Object, e As EventArgs)
    Dim FileName As TextBox = DirectCast(sender, TextBox)
    FileName.Update()
    SavedFileName = FileName.Text()
End Sub
```

```
Private Sub SaveGame_Click(sender As Object, e As EventArgs)
    Dim Save As IO.StreamWriter
    Save = New IO.StreamWriter(SavedFileName & ".txt")
    Save.WriteLine(BoardSize)
    Save.WriteLine(NumberOfplayers)
    Save.WriteLine(Difficulty)
    Save.WriteLine(CurrentPlayer)
    Save.WriteLine(GameNumber)
    Save.WriteLine(NoOfGames)
    For i = 0 To WinnerOfGames.Count - 1
        Save.Write(WinnerOfGames(i))
    Next
    Save.WriteLine()
    For Column = 1 To BoardSize
        For Row As Integer = 1 To BoardSize
            Save.Write(BoardState(Column, Row))
            If BoardState(Column, Row) = Nothing Then
                Save.Write("E")
            End If
        Next
        Save.WriteLine()
    Next
    Save.Close()
End Sub

Private Sub Board_click(sender As Object, e As EventArgs)
    'This sub handles when one of the butttons on the board is clicked
    Dim ButtonClicked As Button = DirectCast(sender, Button)
    If Not (ButtonClicked.BackColor = Color.Red Or ButtonClicked.BackColor = Color.Blue Or
CurrentPlayer = "N") Then
```

```
        'This if statemnet checks if the button was already pressed so you cant overwrite someone
elses move

        For Column = 1 To BoardSize
          For Row As Integer = 1 To BoardSize
            If ButtonClicked.Location = New Point(Row * 100 + 300 + Column * 50, 100 + Column *
85) Then
              BoardState(Column, Row) = CurrentPlayer
              'Determines which button was pressed so you can set the board state equal to the
player who pressed it
            End If
          Next
        Next
        If CurrentPlayer = "R" Then
          ButtonClicked.BackColor = Color.Red
        Else
          ButtonClicked.BackColor = Color.Blue
        End If
        'This Changes the colour of the button
        If (NumberOfplayers = 2 Or NumberOfplayers = 1 And CurrentPlayer = "R") And UsePiRule =
True Then
          Dim Switch As New Button
          Switch.Location = New Point(800, 800)
          Switch.Size = New Size(130, 90)
          Switch.Font = New Font("Comic Sans MS", 18)
          Switch.Text = "Switch"
          Switch.Name = "Switch"
          AddHandler Switch.Click, AddressOf Switch_Click
          Me.Controls.Add(Switch)
          Dim Stay As New Button
          Stay.Location = New Point(500, 800)
          Stay.Size = New Size(130, 90)
```

```
            Stay.Font = New Font("Comic Sans MS", 18)

            Stay.Text = "Stay"

            Stay.Name = "Stay"

            AddHandler Stay.Click, AddressOf Stay_Click

            Me.Controls.Add(Stay)

            UsePiRule = False

        End If

        If NumberOfplayers = 1 And UsePiRule = True Then

            UsePiRule = False

            If ButtonClicked.Name.Contains(BoardSize) Or ButtonClicked.Name.Contains(1) Or
ButtonClicked.Name.Contains(BoardSize - 1) Or ButtonClicked.Name.Contains(2) Then

                MessageBox.Show("Computer chose stay")

            Else

                MessageBox.Show("Computer chose switch")

                CurrentPlayer = "R"

                Exit Sub

            End If

        End If

        MoveAftermath()


    ElseIf CurrentPlayer = "N" Then

        MessageBox.Show("Game Over")

    Else

        MessageBox.Show("Already clicked")

    End If

  End Sub


  Sub MoveAftermath()


    'This decides what happens after a button is pressed

    Dim Nextlist(BoardSize) As String
```

```
        'Nextlist is the current row(Column if blue) you are on

        Dim Whichlist(BoardSize) As String

        'Whichlist is the next row(Column if blue)

        Dim Column As Integer = 1

        For i = 1 To BoardSize

            WinCheck(i, Whichlist, Nextlist, Column)

            ' This checks whether anyone has won yet

            Column = 1


            If CurrentPlayer = "N" Then

                'When Currentplayer is N that means someone has won so this calls the sub that will reset
the game

                If GameNumber < NoOfGames Then

                    GameNumber = GameNumber + 1

                    Me.Controls.Clear()

                    ReDim BoardState(BoardSize, BoardSize)

                    CurrentPlayer = "B"

                    MakeBoard()

                Else

                    GameNumber = 1

                    ResetGame()

                End If

                Exit Sub

            End If


        Next

        If CurrentPlayer = "R" Then

            CurrentPlayer = "B"

        Else

            CurrentPlayer = "R"

        End If
        'Above changes the player to the next player
        If ComputersGo = True Then
            If NumberOfplayers = 1 Then
                'This checks if the game is two player or one player and if its one player calls the sub
according to the difficulty
                If Difficulty = 3 Then
                    Difficulty3()
                End If
                If Difficulty = 2 Then
                    Difficulty2()
                End If
                If Difficulty = 1 Then
                    Difficulty1()
                End If
                'ReDim Pathlength(BoardSize, BoardSize)
            End If
            ComputersGo = True
            'This will stop the computer pressing a button looping
        End If
    End Sub

    Sub ResetGame()
        If NoOfGames <> 1 Then
            Dim SeriesWinner As Integer
            For i = 0 To WinnerOfGames.Count - 1
                If WinnerOfGames(i) = "R" Then
                    SeriesWinner = SeriesWinner + 1
                Else
                    SeriesWinner = SeriesWinner - 1
                End If
            Next
            If SeriesWinner > 0 Then
                MessageBox.Show("Red won the series")
            ElseIf SeriesWinner = 0 Then
                MessageBox.Show("Series was a draw")
            Else
                MessageBox.Show("Blue won the series")
            End If
        End If



        'Creates the buttons to reset the game
        Dim TextForPlayAgain As New Label
        TextForPlayAgain.Location = New Point(200, 300)
        TextForPlayAgain.Size = New Size(300, 50)
        TextForPlayAgain.Font = New Font("Comic Sans MS", 18, FontStyle.Bold Or FontStyle.Underline)
        TextForPlayAgain.Text = "Play Again"
        Me.Controls.Add(TextForPlayAgain)


        Dim PlayAgainYes As New Button
        PlayAgainYes.Location = New Point(150, 400)
        PlayAgainYes.Size = New Size(100, 50)
        PlayAgainYes.Font = New Font("Comic Sans MS", 14)
        PlayAgainYes.Text = "Yes"
        AddHandler PlayAgainYes.Click, AddressOf Yes_click
        Me.Controls.Add(PlayAgainYes)


        Dim PlayAgainNo As New Button
```

```
        PlayAgainNo.Location = New Point(300, 400)

        PlayAgainNo.Size = New Size(100, 50)

        PlayAgainNo.Font = New Font("Comic Sans MS", 14)

        PlayAgainNo.Text = "No"

        AddHandler PlayAgainNo.Click, AddressOf No_click

        Me.Controls.Add(PlayAgainNo)


    End Sub


    Private Sub Yes_click(sender As Object, e As EventArgs)

        'If they click play agian this clears board and resets the boardstate back to the beginning. it then
calls the sub that creates the menu again

        Dim ButtonClicked As Button = DirectCast(sender, Button)

        Me.Controls.Clear()

        ReDim BoardState(BoardSize, BoardSize)

        CreateMenu()

        CurrentPlayer = "B"

    End Sub

    Private Sub No_click(sender As Object, e As EventArgs)

        'This will close the form if they dont want to play again

        Dim ButtonClicked As Button = DirectCast(sender, Button)

        Close()

    End Sub




    Sub CreateRowOrColumn(ByRef Whichlist, ByRef NextList, ByRef Column)

        If CurrentPlayer = "R" Then

            'This finds the current row and next row you are on and sets each element equal to the
equatible element in boardstate

            For i = 1 To BoardSize

                Whichlist(i) = BoardState(Column, i)

                NextList(i) = BoardState(Column + 1, i)
            Next

        ElseIf CurrentPlayer = "B" Then

            'This finds the current Column and next column you are on and sets each element equal to
the equatible element in boardstate

            For i = 1 To BoardSize


                Whichlist(i) = BoardState(i, Column)

                NextList(i) = BoardState(i, Column + 1)

            Next

        End If

    End Sub


    Public Sub WinCheck(ByVal Row, ByVal Whichlist, ByVal Nextlist, ByVal Column)


        If CurrentPlayer = "R" Or CurrentPlayer = "B" Then

            CreateRowOrColumn(Whichlist, Nextlist, Column)

            'The call above gets the current and next row or column

            If Whichlist(Row) = CurrentPlayer Then

                Column = Column + 1

                If Whichlist(Row) = Nextlist(Row - 1) Or Whichlist(Row) = Nextlist(Row) Then

                    'Sees if the tile in the same row and below is the same as the one your on

                    If Whichlist(Row) = Nextlist(Row - 1) Then


                        If Column >= BoardSize Then

                            DisplayWinner()

                            'This tells if there is a complete path from top to bottom or left to right

                        End If


                        WinCheck(Row - 1, Whichlist, Nextlist, Column)


                        'Calls this sub again using the new value of what tile you are on
                    ElseIf Whichlist(Row) = Nextlist(Row) Then
                        If Column >= BoardSize Then
                            DisplayWinner()
                        End If
                        WinCheck(Row, Whichlist, Nextlist, Column)
                    End If
                End If


            End If
            If Row <> BoardSize Then
                'Checks to the left and right of the current tile
                'Makes it so the program doesnt break by having it be out of the array
                If (Whichlist(Row) = Whichlist(Row + 1) Or Whichlist(Row) = Whichlist(Row - 1)) And 
Whichlist(Row) = CurrentPlayer Then
                    If Whichlist(Row) = Whichlist(Row + 1) Then
                        Column = Column - 1
                        WinCheck(Row + 1, Whichlist, Nextlist, Column)



                    End If
                    'Calls this sub again if the tile to the right of the current tile is the same colour
                    If Whichlist(Row) = Whichlist(Row - 1) Then
                        Column = Column - 1
                        Checkleft(Row - 1, Whichlist, Column)
                        'Calls the checkleft sub if the tile to the left is the same colour as the current tile
                    End If


                    End If



            End If
```

```
        End If

        'Whichlist(Row) = "D"

        'Column = Column - 1

    End Sub

    Sub Checkleft(ByVal Row, ByVal Whichlist, ByVal Column)

        Dim Nextlist(BoardSize) As String

        If CurrentPlayer = "R" Or CurrentPlayer = "B" Then

            CreateRowOrColumn(Whichlist, Nextlist, Column)

            'This sub does nearly the same as WinCheck but this doesnt check right so it doesnt cause an
infinite loop of checking right and left


            Column = Column + 1

            If Whichlist(Row) = "R" Or Whichlist(Row) = "B" Then


                If Whichlist(Row) = Nextlist(Row - 1) Or Whichlist(Row) = Nextlist(Row) Then

                    If Whichlist(Row) = Nextlist(Row - 1) Then


                        If Column >= BoardSize Then

                            DisplayWinner()

                            'This tells if there is a complete path from top to bottom or left to right so if someone
has won

                        End If


                        WinCheck(Row - 1, Whichlist, Nextlist, Column)

                    ElseIf Whichlist(Row) = Nextlist(Row) Then

                        If Column >= BoardSize Then

                            DisplayWinner()

                        End If

                        WinCheck(Row, Whichlist, Nextlist, Column)

                    End If

                End If
            End If

            If Row > 0 Then


                If Whichlist(Row) = Whichlist(Row - 1) Then

                    Column = Column - 1

                    Checkleft(Row - 1, Whichlist, Column)

                    'Calls check left again if the left tile is again the same as the current tile

                End If

            End If

        End If


    End Sub

    Sub DisplayWinner()

        If CurrentPlayer = "R" Then

            WinnerOfGames(GameNumber - 1) = "R"

            MessageBox.Show("Red wins")

        Else

            WinnerOfGames(GameNumber - 1) = "B"

            MessageBox.Show("Blue wins")

        End If

        CurrentPlayer = "N"

    End Sub

    Sub ComputerPlayer(ByRef CoordOnPath)

        'This sets boardgraph and then calls djikstra algortihm

        If NumberOfplayers = 1 Then

            SetBoardGarph()

            Dim FindPath As New path(BoardSize, BoardGraph)

            FindPath.SetupDjikstra(CoordOnPath)

        End If


    End Sub
```

```
  Sub Difficulty3()
      Dim GeneralValue(BoardSize, BoardSize) As Integer
      Dim CoordOnPathHold1 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer,
Integer))
      Dim CoordOnPathHold2 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer,
Integer))
      Dim HighestMoveValue As Integer
      Dim UniqueCoords(BoardSize, BoardSize) As Integer
      Dim CoordTuple As Tuple(Of Integer, Integer)
      Dim FinalList As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
      Dim RandomCoord As Integer
      If CurrentPlayer = "R" Then
        CurrentPlayer = "B"
      Else
        CurrentPlayer = "R"
      End If
      'Changes player so it does a djikstra for the oponent and stores the shortest paths in
CoorOnPathHold1
      ComputerPlayer(CoordOnPathHold1)
      'CoordOnPathHold1 = FlipBoard(CoordOnPathHold1)


      FindGoodMove(CoordOnPathHold1, UniqueCoords)
      'Calls FindGoodMove sub
      If CurrentPlayer = "R" Then
        CurrentPlayer = "B"
      Else
        CurrentPlayer = "R"


      End If
      'Chnages player so it does a djikstra for itself and stores the shortest paths in CoorOnPathHold2
      ComputerPlayer(CoordOnPathHold2)
```

```
        CoordOnPathHold2 = FlipBoard(CoordOnPathHold2)

        'Flips the board so that board orietation is the same for both

        FindGoodMove(CoordOnPathHold2, UniqueCoords)


        GeneralValue = BasicMoveValues(GeneralValue)


        For column = 1 To BoardSize

           For row = 1 To BoardSize

              UniqueCoords(column, row) = UniqueCoords(column, row) + GeneralValue(column, row)

           Next

        Next


        'Combines the djikstras and the values of coords around the board to find highest value  of all
spaces on the board

        For column = 1 To BoardSize

           For Row = 1 To BoardSize

              If UniqueCoords(column, Row) > HighestMoveValue Then

                 HighestMoveValue = UniqueCoords(column, Row)

              End If

           Next

        Next

        'Above finds the highest value of all tiles on board

        For column = 1 To BoardSize

           For Row = 1 To BoardSize

              If UniqueCoords(column, Row) = HighestMoveValue Then

                 CoordTuple = New Tuple(Of Integer, Integer)(column, Row)

                 FinalList.Add(CoordTuple)

                 'All coords with the highest value are added to a list to find all best moves

              End If

           Next

        Next
        'Gets one set of coords from the sub
        If CurrentPlayer = "R" Then
            ClickComputerMove(FinalList(RandomCoord).Item2, FinalList(RandomCoord).Item1)
        Else
            ClickComputerMove(FinalList(RandomCoord).Item1, FinalList(RandomCoord).Item2)
        End If
        'Calls sub that actually clicks the button
    End Sub
    Sub FindGoodMove(ByVal CoordOnPathHold, ByRef UniqueCoords)
        For column = 1 To BoardSize
            For Row = 1 To BoardSize
                For FindOverlap = 0 To CoordOnPathHold.Count - 1
                    If CoordOnPathHold(FindOverlap) IsNot Nothing Then
                        If CoordOnPathHold(FindOverlap).Item1 = column And
CoordOnPathHold(FindOverlap).Item2 = Row Then
                            UniqueCoords(column, Row) = UniqueCoords(column, Row) + 1
                        End If
                    End If
                Next
            Next
        Next
        'Finds overlap of points to see which points come up where on each djikstra
    End Sub
    Function FindMoveInList(CoordOnPathHold)
        Dim RandomCoord As Integer
        Dim ValidCoord As Boolean = False
        Randomize()

        Do
```

```
            RandomCoord = Int((CoordOnPathHold.Count - 1) * Rnd())


            If CoordOnPathHold(RandomCoord) IsNot Nothing Then
                If BoardGraph(CoordOnPathHold(RandomCoord).Item1,
CoordOnPathHold(RandomCoord).Item2) = 1 Then
                    ValidCoord = True
                End If
            End If
        Loop Until ValidCoord = True
        'Randomly selects a coord from the list of coords put in
        Return RandomCoord
    End Function
    Function FlipBoard(CoordOnPathHold)
        If CurrentPlayer = "B" Then
            For Reverse = 0 To CoordOnPathHold.Count - 1
                If CoordOnPathHold(Reverse) IsNot Nothing Then
                    CoordOnPathHold(Reverse) = New Tuple(Of Integer,
Integer)(CoordOnPathHold(Reverse).Item2, CoordOnPathHold(Reverse).Item1)
                End If
            Next
        End If
        'this flips the columns and rows for the djikstra so that the rows and columns are the same for
both djikstras
        Return CoordOnPathHold
    End Function


    Sub Difficulty2()
        Dim FinalList As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
        Dim GeneralValue(BoardSize, BoardSize) As Integer
        Dim RandomCoord As Integer
        Dim CoordOnPathHold1 As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer,
Integer))
```

```
Dim HighestMoveValue As Integer
Dim CoordTuple As Tuple(Of Integer, Integer)
ComputerPlayer(CoordOnPathHold1)
CoordOnPathHold1 = FlipBoard(CoordOnPathHold1)
BasicMoveValues(GeneralValue)
FindGoodMove(CoordOnPathHold1, GeneralValue)
RandomCoord = FindMoveInList(CoordOnPathHold1)
'Does djikstras once and then finds a random coord on the shortest path then has it clicked
For column = 1 To BoardSize
  For Row = 1 To BoardSize
    If GeneralValue(column, Row) > HighestMoveValue Then
      HighestMoveValue = GeneralValue(column, Row)
    End If
  Next
Next
'Above finds the highest value of all tiles on board
For column = 1 To BoardSize
  For Row = 1 To BoardSize
    If GeneralValue(column, Row) = HighestMoveValue Then
      CoordTuple = New Tuple(Of Integer, Integer)(column, Row)
      FinalList.Add(CoordTuple)
      'All coords with the highest value are added to a list to find all best moves
    End If
  Next
Next
RandomCoord = FindMoveInList(FinalList)
'Gets one set of coords from the sub
If CurrentPlayer = "R" Then
  ClickComputerMove(FinalList(RandomCoord).Item2, FinalList(RandomCoord).Item1)
Else
  ClickComputerMove(FinalList(RandomCoord).Item1, FinalList(RandomCoord).Item2)
```

```
        End If

        'Calls sub that actually clicks the button

    End Sub

    Sub Difficulty1()

        Dim RandomXCoord As Integer

        Dim RandomYCoord As Integer

        SetBoardGarph()

        Do

            Randomize()

            RandomXCoord = Int((BoardSize) * Rnd() + 1)

            RandomYCoord = Int((BoardSize) * Rnd() + 1)

        Loop Until BoardGraph(RandomYCoord, RandomXCoord) = 1

        ClickComputerMove(RandomXCoord, RandomYCoord)

        'Finds a random coord that isnt pressed on the board and then presses it

    End Sub

    Sub ClickComputerMove(ByVal XCoord, ByVal YCoord)

        ComputersGo = False

        Board = FindFromButtons(YCoord & XCoord)

        Board.PerformClick()

        'Finds the button with the name of the coords that are chosen and then sets a button equal to
that and clicks it

    End Sub


    Sub SetBoardGarph()

        If CurrentPlayer = "R" Then


            For P = 0 To BoardSize

                For y = 0 To BoardSize

                    If CurrentPlayer = BoardState(P, y) Then

                        BoardGraph(P, y) = 0

                    ElseIf P = 0 Or y = 0 Then
                        BoardGraph(P, y) = 9999999999999
                    ElseIf BoardState(P, y) = "" Then
                        BoardGraph(P, y) = 1
                    Else
                        BoardGraph(P, y) = 1000
                    End If
                Next
            Next
        End If
        'Above finds the values of the tiles in the way we look at it
        If CurrentPlayer = "B" Then

            For y = 0 To BoardSize
                For p = 0 To BoardSize
                    If CurrentPlayer = BoardState(p, y) Then
                        BoardGraph(y, p) = 0
                    ElseIf p = 0 Or y = 0 Then
                        BoardGraph(y, p) = 9999999999999
                    ElseIf BoardState(p, y) = "" Then
                        BoardGraph(y, p) = 1
                    Else
                        BoardGraph(y, p) = 1000
                    End If
                Next
            Next
        End If
        'Above finds the values of the tiles rotated 90 degrees so the first column is now the top row
    End Sub
```

```
Function BasicMoveValues(ByRef GeneralValue)
    For column = 1 To BoardSize
        For Row = 1 To BoardSize
            If BoardGraph(column, Row) <> 1 Then
                BoardGraph(column, Row) = -1000
                Try
                    If BoardGraph(column + 2, Row) = 1 Then
                        GeneralValue(column + 2, Row) = 3 + BoardGraph(column + 2, Row)
                        If BoardGraph(column + 1, Row) = 1 Then
                            GeneralValue(column + 1, Row) = 2 + BoardGraph(column + 1, Row)
                        End If
                        If BoardGraph(column, Row + 1) = 1 Then
                            GeneralValue(column, Row + 1) = 2 + BoardGraph(column, Row + 1)
                        End If
                    End If
                Catch ex As Exception

                End Try
                Try
                    If BoardGraph(column + 1, Row + 1) = 1 Then
                        GeneralValue(column + 1, Row + 1) = 3 + BoardGraph(column + 1, Row + 1)
                        If BoardGraph(column + 1, Row) = 1 Then
                            GeneralValue(column + 1, Row) = 2 + BoardGraph(column + 1, Row)
                        End If
                        If BoardGraph(column, Row + 1) = 1 Then
                            GeneralValue(column, Row + 1) = 2 + BoardGraph(column, Row + 1)
                        End If
                    End If
```

```
Catch ex As Exception


End Try
Try
    If BoardGraph(column - 1, Row + 2) = 1 Then
        GeneralValue(column - 1, Row + 2) = 3 + BoardGraph(column - 1, Row + 2)
        If BoardGraph(column - 1, Row) = 1 Then
            GeneralValue(column - 1, Row) = 2 + BoardGraph(column - 1, Row)
        End If
        If BoardGraph(column, Row + 1) = 1 Then
            GeneralValue(column, Row + 1) = 2 + BoardGraph(column, Row + 1)
        End If
    End If
Catch ex As Exception


End Try


Try
    If BoardGraph(column - 1, Row - 1) = 1 Then
        GeneralValue(column - 1, Row - 1) = 3 + BoardGraph(column - 1, Row - 1)
        If BoardGraph(column - 1, Row) = 1 Then
            GeneralValue(column - 1, Row) = 2 + BoardGraph(column - 1, Row)
        End If
        If BoardGraph(column, Row - 1) = 1 Then
            GeneralValue(column, Row - 1) = 2 + BoardGraph(column, Row - 1)
        End If
    End If
Catch ex As Exception


End Try
```

```
                Try
                    If BoardGraph(column + 1, Row - 2) = 1 Then
                        GeneralValue(column + 1, Row - 2) = 3 + BoardGraph(column + 1, Row - 2)
                        If BoardGraph(column - 1, Row) = 1 Then
                            GeneralValue(column - 1, Row) = 2 + BoardGraph(column - 1, Row)
                        End If
                        If BoardGraph(column, Row - 1) = 1 Then
                            GeneralValue(column, Row - 1) = 2 + BoardGraph(column, Row - 1)
                        End If
                    End If
                Catch ex As Exception

                End Try
            End If

        Next
    Next
    'Finds general good values of moves as they are generally just good to make
    Return GeneralValue
End Function



End Class


Class path
    Dim Pathlength(boardsize, boardsize) As Integer
    Dim ShortestValue As Integer = 10000000
    Dim ShortestPoint As Tuple(Of Integer, Integer)
    Dim ValidPath As Boolean = False
    Dim Found As List(Of Tuple(Of Integer, Integer))
    Dim Paths(boardsize, boardsize) As Tuple(Of Integer, Integer)

    Dim Coords As Tuple(Of Integer, Integer)

    Dim LastPoint As Tuple(Of Integer, Integer)

    Dim DijkstraLeft As Boolean

    Dim DijkstraRight As Boolean

    Dim HoldLastRow(boardsize) As Integer

    Dim boardsize As Integer

    Dim boardgraph(boardsize, boardsize)

    Sub New(ByVal boardsize As Integer, ByVal boardgraph(,) As Object)

        Me.boardsize = boardsize

        Me.boardgraph = boardgraph

        ReDim Paths(boardsize, boardsize)

        ReDim Pathlength(boardsize, boardsize)

        ReDim HoldLastRow(boardsize)


    End Sub

    Sub SetupDjikstra(ByRef CoordOnPath)

        If Found Is Nothing Then

            Found = New List(Of Tuple(Of Integer, Integer))


            'This initializes found so that you can add items to the list

        End If


        For FirstRow = 1 To boardsize

            Pathlength(1, FirstRow) = boardgraph(1, FirstRow)


            Found.Add(New Tuple(Of Integer, Integer)(1, FirstRow))

        Next

        'Above for loop gets all the coordinates of the first row or column depending on the current
  player

        For FindLastRow = 1 To boardsize
            HoldLastRow(FindLastRow) = 0

            'Finds the value of the last row for a later check to see if all the paths have been found

        Next

        While ValidPath = False

            ShortestValue = 100000000

            For CurrentNode = 0 To Found.Count - 1


                Coords = Found.Item(CurrentNode)

                If Coords.Item1 <> boardsize Then


                    If Coords.Item2 <> boardsize And Coords.Item2 >= 1 Then

                        DijkstraLeft = True

                        DijkstraRight = True

                        Dijkstras()

                    ElseIf Coords.Item2 = boardsize Then

                        DijkstraLeft = True

                        DijkstraRight = False

                        Dijkstras()

                    ElseIf Coords.Item2 <= 1 Then

                        DijkstraRight = True

                        DijkstraLeft = False

                        Dijkstras()

                    End If

                End If

                'Sees if the thing can check left or right so that it doesnt crash if the value is out of the array

            Next


            Pathlength(ShortestPoint.Item1, ShortestPoint.Item2) = ShortestValue

            Found.Add(ShortestPoint)

            Paths(ShortestPoint.Item1, ShortestPoint.Item2) = LastPoint

            'Adds the point which is shortest to get to to the list of coords found
```

```
            For PathlengthAllLastRow = 1 To boardsize
                If Pathlength(boardsize, PathlengthAllLastRow) = HoldLastRow(PathlengthAllLastRow) Then
                    ValidPath = False
                    PathlengthAllLastRow = boardsize
                Else
                    ValidPath = True
                End If
                'Checks if all paths to the furhtest point possible has been found
            Next
        End While
        ShortestPathCoords(CoordOnPath)
        'Calls shortestpointsCOords sub
        ValidPath = False
        Found.Clear()
        'Above two lines resets this so it can run again for the next move the computer makes
    End Sub
    Sub Dijkstras()
        If DijkstraLeft = True Then

            If boardgraph(Coords.Item1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2) <
ShortestValue Then
                If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 - 1))
Then
                    ShortestValue = boardgraph(Coords.Item1, Coords.Item2 - 1) + Pathlength(Coords.Item1,
Coords.Item2)
                    ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 - 1))
                    LastPoint = Coords
                End If
            End If
            If boardgraph(Coords.Item1 + 1, Coords.Item2 - 1) + Pathlength(Coords.Item1, Coords.Item2)
< ShortestValue Then
                If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2 - 1))
Then
```

```
                ShortestValue = boardgraph(Coords.Item1 + 1, Coords.Item2 - 1) +
Pathlength(Coords.Item1, Coords.Item2)

                ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2 - 1))

                LastPoint = Coords

            End If

        End If

    End If

    If boardgraph(Coords.Item1 + 1, Coords.Item2) + Pathlength(Coords.Item1, Coords.Item2) <
ShortestValue Then

        If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2)) Then

            ShortestValue = boardgraph(Coords.Item1 + 1, Coords.Item2) + Pathlength(Coords.Item1,
Coords.Item2)

            ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1 + 1, Coords.Item2))

            LastPoint = Coords

        End If

    End If

    If DijkstraRight = True Then

        If boardgraph(Coords.Item1, Coords.Item2 + 1) + Pathlength(Coords.Item1, Coords.Item2) <
ShortestValue Then

            If Not Found.Contains(New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 + 1))
Then

                ShortestValue = boardgraph(Coords.Item1, Coords.Item2 + 1) + Pathlength(Coords.Item1,
Coords.Item2)

                ShortestPoint = (New Tuple(Of Integer, Integer)(Coords.Item1, Coords.Item2 + 1))

                LastPoint = Coords

            End If

        End If

    End If

    'Does djikstras algorithm to find shortest next point to get to is

End Sub


Dim EndPoints As List(Of Tuple(Of Integer, Integer)) = New List(Of Tuple(Of Integer, Integer))
```

```
Sub ShortestPathCoords(ByRef CoordOnPath)
    Dim Pathcount As Integer
    Dim StopRepeat As Integer = 0
    Dim Small As Integer = 1000
    For Column = 0 To boardsize - 1
        If Pathlength(boardsize, boardsize - Column) < Small And Pathlength(boardsize, boardsize - Column) <> 0 Then
            Small = Pathlength(boardsize, boardsize - Column)
        End If
    Next
    'Finds the smallest value of the end column to see which column ends in the lowest amount of moves needed to reach
    For CoordFind = 0 To boardsize - 1
        If Pathlength(boardsize, boardsize - CoordFind) = Small Then
            Pathcount = Pathcount + 1
            CoordOnPath.Add(New Tuple(Of Integer, Integer)(boardsize, boardsize - CoordFind))
        End If
    Next
    EndPoints = CoordOnPath
    For holdname = 0 To EndPoints.Count - 1
        CoordOnPath.Add(Paths(CoordOnPath(holdname).Item1, CoordOnPath(holdname).Item2))
        Do Until CoordOnPath(CoordOnPath.Count - 1) Is Nothing
            CoordOnPath.Add(Paths(CoordOnPath(CoordOnPath.Count - 1).Item1, CoordOnPath(CoordOnPath.Count - 1).Item2))
        Loop
    Next
    'Finds all the coords that are needed for the shortest path and puts them in a list
    ReDim Pathlength(boardsize, boardsize)
End Sub
End Class
```

Shaped Controls

```
Imports System.Runtime.CompilerServices


Module ShapedControls


    Public Const Pi As Double = Math.PI

      Public Const DegreesToRadians As Double = 180 / Pi


      <Extension()>

      Public Sub Shape(ByVal ctrl As Control,

   Optional ByVal NumberOfSides As Integer = 3,

   Optional ByVal OffsetAngleInDegrees As Double = 0)


        If NumberOfSides < 3 Then Throw New Exception("Number of sides can only be 3 or
more.")


        Dim MyAngle As Double = OffsetAngleInDegrees / DegreesToRadians


        Dim radius1 As Integer = ctrl.Height \ 2

        Dim radius2 As Integer = ctrl.Width \ 2

        Dim xInt, yInt As Integer

        Dim xDoub, yDoub As Double

        Dim MyPath As New Drawing2D.GraphicsPath
        For angle As Double = MyAngle To ((2 * Pi) + MyAngle) Step ((2 * Pi) / NumberOfSides)

            xDoub = radius2 * Math.Cos(angle) + radius2
            yDoub = radius1 * Math.Sin(angle) + radius1
            xInt = CInt(Int(xDoub))
            yInt = CInt(Int(yDoub))
            MyPath.AddLine(New Point(xInt, yInt), New Point(xInt, yInt))

        Next

        MyPath.CloseFigure()
        ctrl.Region = New Region(MyPath)
        MyPath.Dispose()

    End Sub

End Module
```