Dots and Boxes

Taren Collyer

Candidate Number: 9665

Centre Number: 64395

Godalming College

Contents

Investigation

Research and Analysis

The investigation I've chosen to carry out is whether I can get an AI to learn a turn-based game and play against a player. I want to make use of artifical intelligence in VB in order to enable the computer to play the game. I've decided on using dots and boxes as the base game, as I feel it's a suitable choice that will effectively allow me to implement an AI. I decided to work on such an investigation as I think it would be fascinating to understand the mechanisms and concept behind artificial intelligence, and ways to implement it into an interactive and relatively unpredictable game setting.

What AI Is and the Different Types

AI, or artificial intelligence, is the development of a computer system which enables it to almost mimic human intelligence, performing tasks such as image recognition and decision making. A very common example of AI would be Google's predictive search – as you begin typing, tailored recommendations appear based on the data Google has collected about you. There are a few different types of AI, including:

• **reactive AI:** the most basic type of AI. There is no memory functionality, meaning they don't "learn" from previous experience. They simply respond to a user input or set of inputs. A classic example of a reactive machine is IBM's Deep Blue, the AI that beat chess grandmaster Garry Kasparov in the 1990s.

• **limited memory AI:** similar to a reactive machine, except this type of AI has the capability of responding based upon past data in order to learn. This is the most common application of present-day AI, including Google Search and also things like chat bots or self-driving cars.

• **theory of mind AI:** this is purely a conceptual, work-in-progress type of AI that isn't yet available. It ideally involves giving the AI a perception of human needs, emotions and thoughts – essentially giving machines the ability to understand and independently interact with humans.

• **self-aware AI:** much like the previous type of AI, self-aware machines are not yet available and still very much a work in progress. As the name suggests, this is a type of machine that will have such a high level of human understanding that it will develop a consciousness and become self-aware. They ideally would be able to have such a good understanding of the human mind and emotions that they themselves would have the ability to evoke emotion, have beliefs and desires and interact with people. This is currently the highest target for AI development.

In the case of my investigation, I have concluded that I hope to create a reactive machine which will be able to make decisions based on the user's input and all the remaining positions available to act as opposition to a human player.

Neural Networks

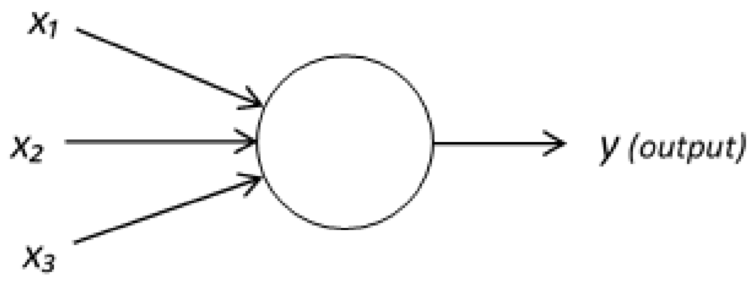
A neural network is essentially a set of algorithms designed to function as closely to the human brain as possible by giving a system the ability to recognise patterns and "learn" how to solve a certain problem. It can learn from past data and become more efficient at working around the problem over time.

Different Parts of a Neural Network

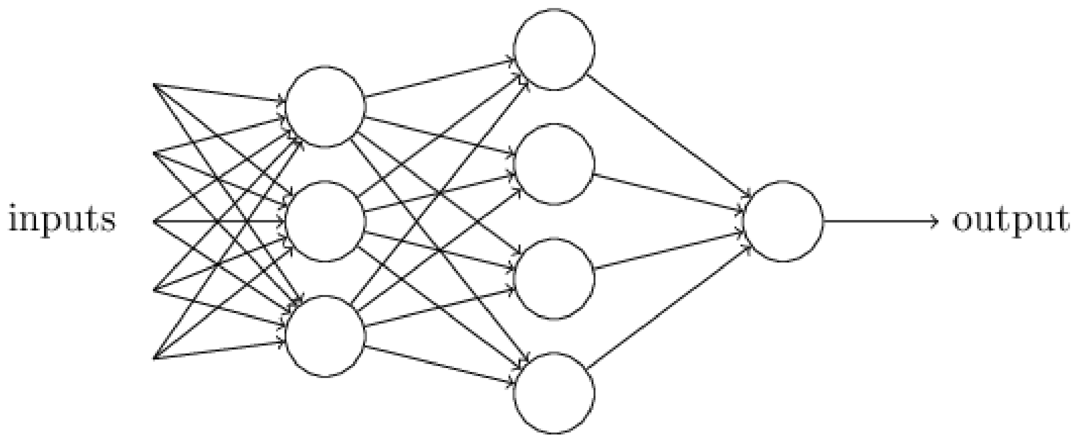
Neural networks are composed of many different artificial neurons, interconnected with each other. There are many different layers of these neurons: the input layer, the hidden layers and the output layer. The input layer of neurons will take the input values and pass them forward into the hidden layers, which are the main bulk of the network responsible for the main learning process, and into the output layers, to generate a final output.

Example of an Artificial Neuron: Perceptrons

An artificial neuron is an essential component of a neural network. It will take an input, process it and as a result produce an output. A perceptron, for example, is a basic type of artificial neuron first developed by American psychologist Frank Rosenblatt in the late 1950s. The basic idea is that the perceptron will take multiple inputs and produce a single output, shown below:



However, Rosenblatt also introduced the idea of numerical weights given to inputs – indicating their importance in relation to the output. This means that the inputs are not merely limited to a binary 1 or 0, and that they can now accept real numerical values. The output is now determined based on what sum of inputs reach a preset threshold value. If, for example, we had 3 inputs with weightings of w_1 = 8, w_2 = 3 and w_3 = 4 and our threshold value was set to 7, we could reach an output of 1 if input 1 (w_1) was true irrelevant of the other inputs as it meets our threshold value, OR if both inputs 2 and 3 (w_2/w_3) were true, as their combined sum would meet our threshold value. In essence, these perceptrons are used in neural networks in order to decisively weigh up evidence and make sophisticated decisions – developing into more intricate networks of different perceptrons to form situations that look more like this:



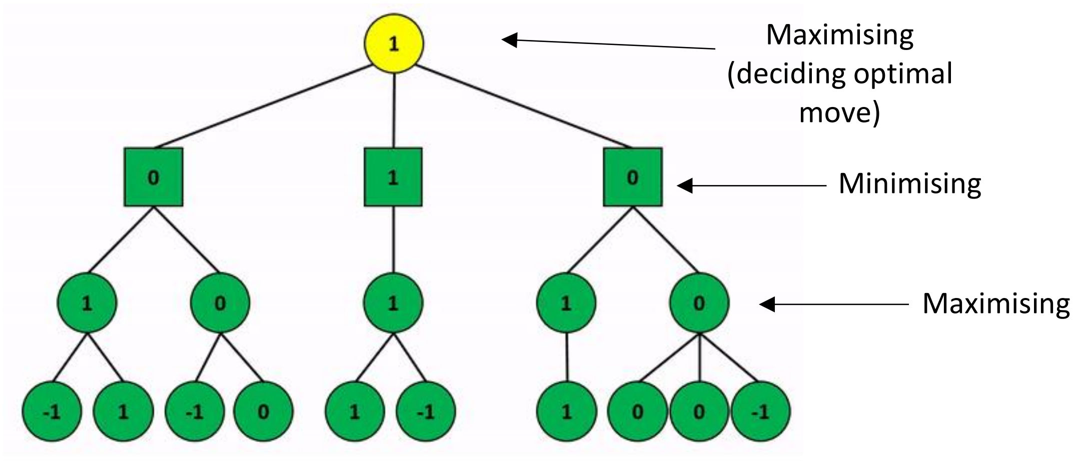
How a Neural Network Learns

Neural networks are made up of various different neurons that are interconnected, much like the network of perceptrons above. Information enters through the input layer of neurons and into the hidden layers, named so due to their nature of being neither an input or an output. After the hidden layers are triggered, the information arrives at the output layers and an output can thus be generated. All of the inputs within a layer are taken from the outputs of the previous layer, in a network type called the feedforward neural network – the simplest neural net design type. The weights of the connections are combined with the inputs received, and, as described in the basic perceptron model, trigger the next layer of neurons if the previous sum reaches the set threshold value. Then, using a process known as back propagation, the system can compare the final output generated in the neural net with the target output, and work out the error margin between the two in order to adjust the weightings between layers going from the output layer back to the input layer – causing the network to develop itself and learn. Its actual output and the target output, over time, eventually will match, meaning the neural net has trained itself and will acquire the desired output.

Minimax

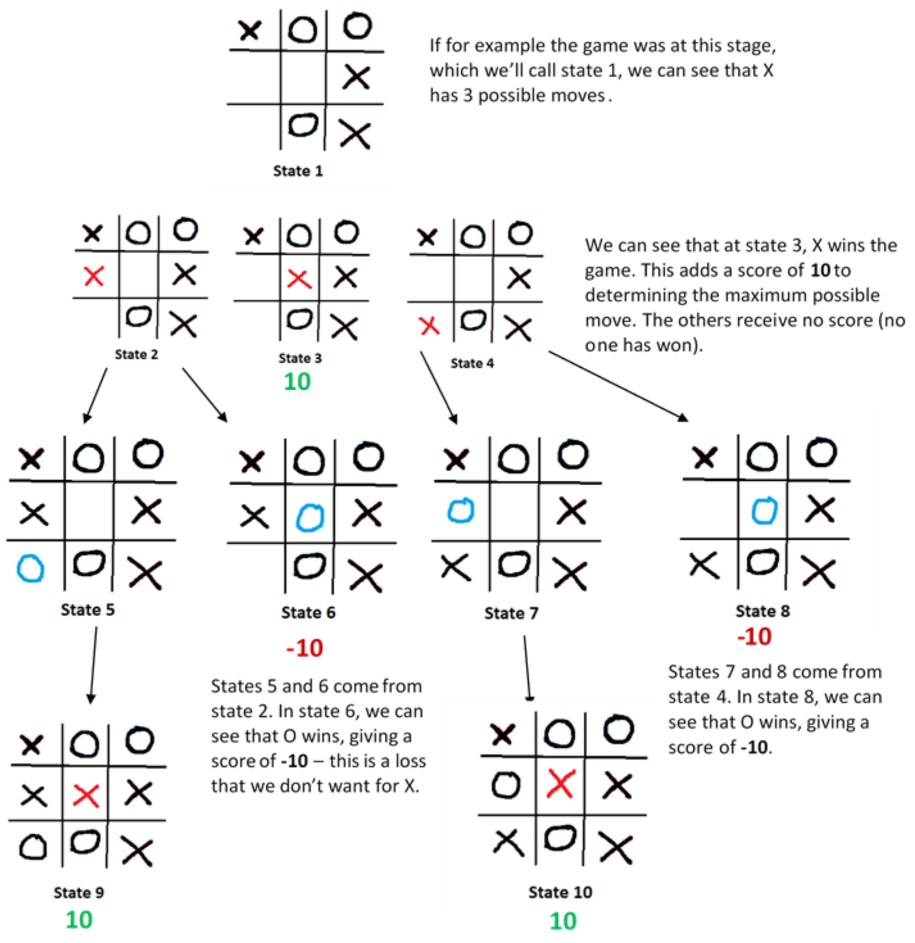
Minimax is a type of algorithm used widely in virtual two-player turn-based games such as the likes of tic tac toe and chess. It is one of the oldest forms of artificial intelligence technology. It acts as a decision making algorithm, its name coming from minimizing the loss when the opposite (human) player takes a turn that would cause the maximum amount of loss.

One player acts as the maximiser, and the other as the minimiser – one finds a game state with the maximum possible score, and the other with the minimum. This means the maximiser tries to get the best possible score while the minimiser attempts to get the lowest score by countering moves. The maximiser will pick the move with the best possible outcome.



Example

In a simple game of tic tac toe, this is a simple explanation of how a minimax could be applied for the AI to work out where to move.



If for example the game was at this stage, which we'll call state 1, we can see that X has 3 possible moves.

We can see that at state 3, X wins the game. This adds a score of **10** to determining the maximum possible move. The others receive no score (no one has won).

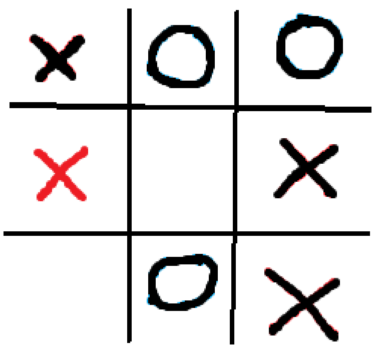
States 5 and 6 come from state 2. In state 6, we can see that O wins, giving a score of **-10** – this is a loss that we don't want for X.

States 7 and 8 come from state 4. In state 8, we can see that O wins, giving a score of **-10**.

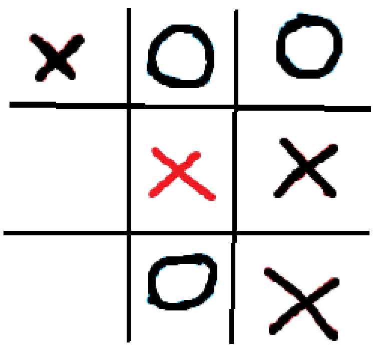
O is the minimiser and X is the maximiser in the above situation. O picks between a score of **+10** (state 5-9) and **-10** (state 6) – giving state 2's possibilities an overall score of **-10** (it wants to get the **biggest loss** out of its options).

It also gives state 4's overall score **-10** due to the choice between **+10** and **-10** (state 4-7 and state 8).

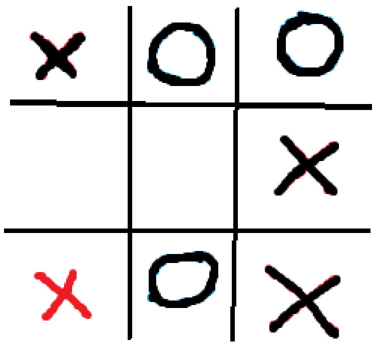
State 3 gets an overall score of **+10** as here it is X's turn and as the maximiser, this allows it to win.


State 2

-10


State 3

10


State 4

-10

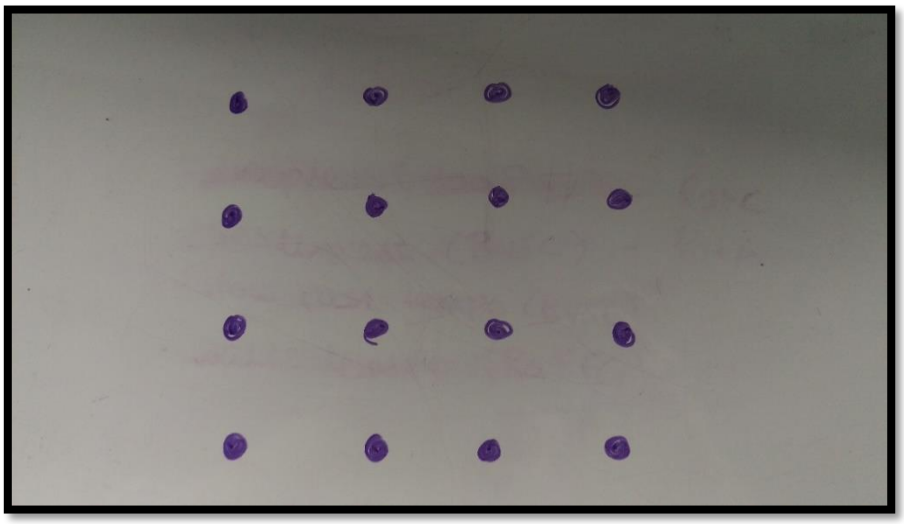
So overall, the best possible move to make in X's case is the position at state 3, as this is the best chance of a win.

Description of Dots and Boxes and How the Game Works

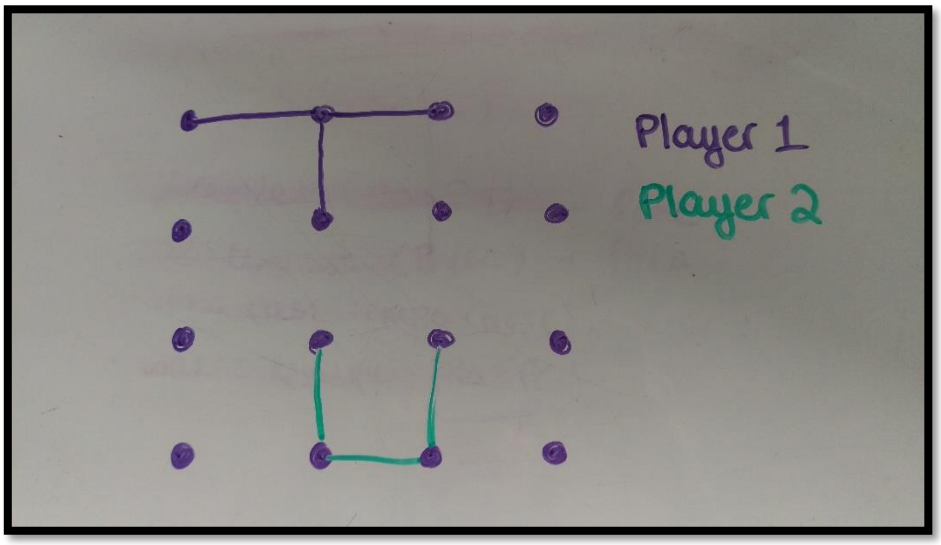
Dots and boxes is a classic game played using pen and paper by 2 players. Its roots date back to as early as the 19th century and has always been a game to revolve around 2 human players, which is why I intend to investigate into whether I can get an AI to compete with the player.

The main goal of the game is to complete as many boxes as you can in your player's colour until the grid is completely filled. The person with the most boxes wins.

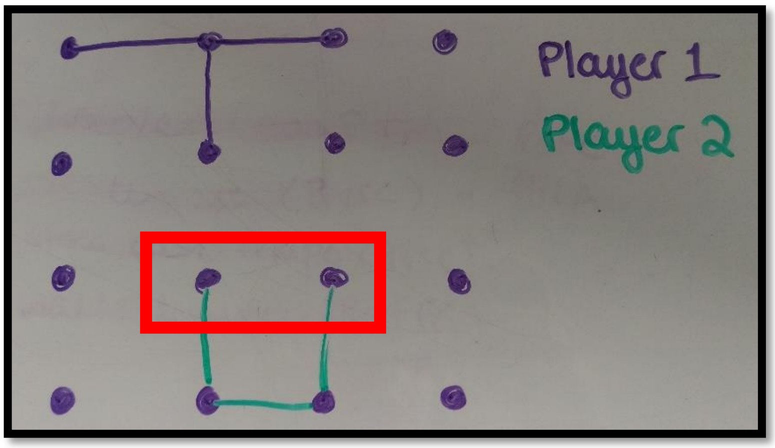
One player starts by placing a single line on a grid of a certain number of dots (usually 4x4) like this one:



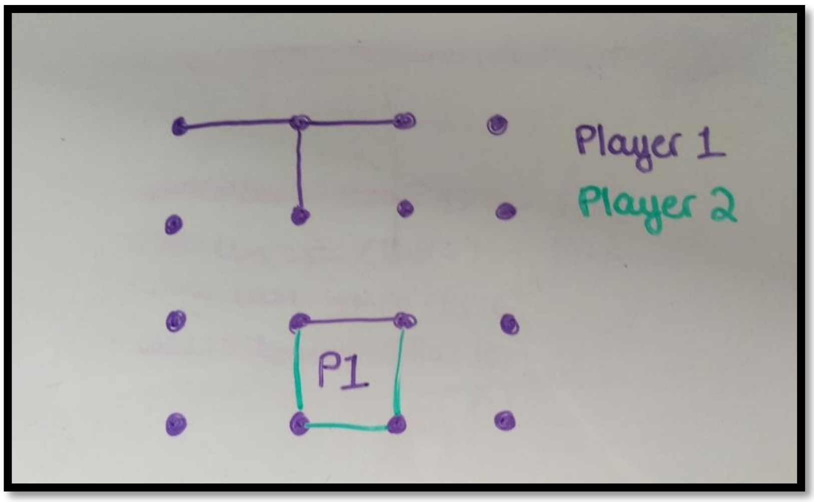
The other player then places a single line anywhere one the grid, in an attempt to make the first player almost complete a 1x1 box. The grid may look something like this after 3 complete turns:



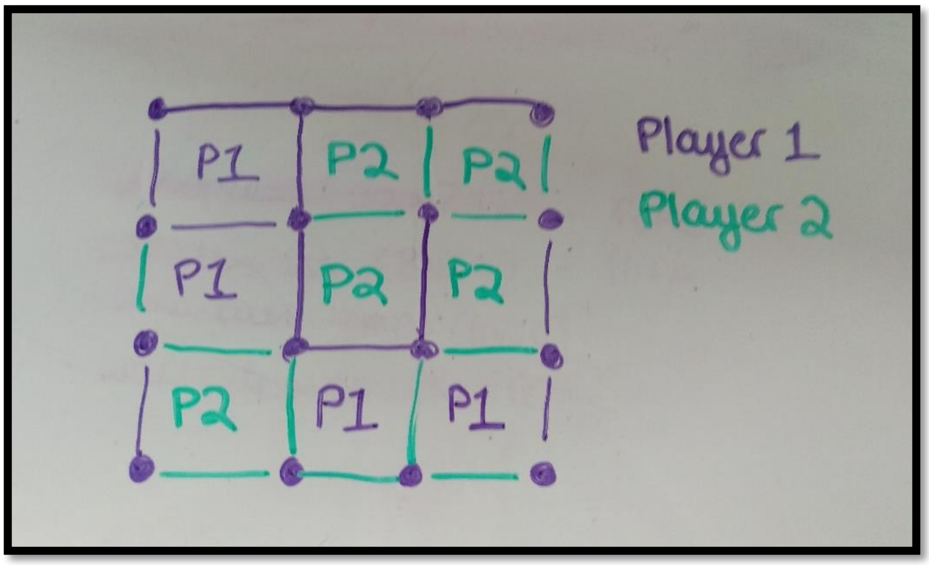
In this situation, both players have taken 3 turns each and player 1 has to start the 4th turn. Player 1 can easily place a line here:



in order to form a 1x1 box and claim it on the grid.



Player 1 can then take another turn after claiming this box.
The game continues in this fashion until the grid is filled with boxes.



The boxes are then counted to see who wins. In this game, since player 2 has 5 boxes and player 1 has 3, player 2 wins.

Using this knowledge and my research on both neural networks and the minimax algorithm, I have concluded that I will need to use a minimax for the AI player as I feel it is most appropriate – it does not need to have the complexity of learning to play the game itself, as this is inappropriate and excessive for a game like this, and merely needs to take into account all possible moves that will enable it to win.

Dialogue With 3rd Party

I spoke with a friend who has an interest in turn-based board games and interviewed her with a few basic questions revolving around the game and turn-based games in general to get a feel for the features the AI could have as well as the game itself. The questions and their results are as follows:

1) When you play dots and boxes, how do you expect the game to work, roughly?
You want to have a player draw a 4 by 4 grid of dots on some paper. It's for 2 players and both need to take turns to put a line between any 2 of the dots on the grid until someone makes a box, which they mark with their initials or a colour or something. When the grid gets filled the game's over and you count up who has the most boxes, then that person wins.

2) And when you're playing, do you think it's important to be able to tell each player's moves apart in a game like this? If so, what do you think the simplest way to show that is?
For this particular game it's not essential, but from my perspective as a player I probably want to know who's moved where, to get a feel for who might or might not win and just see each other's progress. Maybe each player could have their own colour for the lines they draw to show who's who, that would be a good way of showing it.

3) Would it be more helpful if dots and boxes had a score count for the claimed boxes for each player in a computerised version of the game displayed on screen, rather than having to count the boxes?
That would be easier from a player's point of view, yes. Helpful to see who has the advantage at any point in the game.

4) So if an AI is implemented to the game as the second player, how would you want it to behave, in terms of being as "human" as possible? Would different difficulty levels be appropriate?
You'd want it to not be perfect. You wouldn't want it to play perfectly because that's not realistic and not fun. I think different levels would be good, beginners might like an easier computer player while it would be more challenging to have a normal or hard difficulty like other modern computer games.

Upon carrying out this interview, this has given me a more thorough understanding and confirmation of some of the features that I will need to add to my project as part of this investigation in order to satisfy the needs of an average player of dots and boxes and turn-based games as a whole.

Reading Review

• http://theconversation.com/understanding-the-four-types-of-ai-from-reactive-robots-to-self-aware-beings-67616 – this article gave me a really firm grasp and understanding of the four fundamental types of artifical intelligence. It also cites many resources throughout the article based on the information it gives.

• https://learn.g2.com/types-of-artificial-intelligence – I also read through this similar article to confirm my knowledge on the different types of AI.

• https://social.technet.microsoft.com/wiki/contents/articles/32140.basis-of-neural-networks-in-visual-basic-net.aspx – this page on the official Microsoft TechNet site was very insightful into looking at the basis of neural networks and how artificial neurons work and how they essentially simulate natural neurons in a living creature. It detailed methods like backpropogation and showed examples of preparing a sample neural net in Visual Basic.

• https://www.explainthatstuff.com/introduction-to-neural-networks.html – this article also gave me some additional information on the workings and concept behind a neural network, detailing the basics of a simple feedforward neural network.

• https://www.baeldung.com/java-minimax-algorithm – this site gave me a great headstart learning about how the minimax algorithm works. It covered the basics of the maximiser and minimiser components well, but I still felt I needed explained examples to get a better understanding.

• https://www.globalsoftwaresupport.com/minimax-algorithm-explained/ – this article was great in the example it gave, which I later used as inspiration to model my own demonstration of how a minimax algorithm works with tic tac toe. The example was a great help in understanding how the algorithm would work in a real scenario, and it was at this point in my research that I began to feel that this would be the best algorithm to use over a neural network for my particular investigation.

• https://www.youtube.com/watch?v=FLNPAKBJavY – finally, this YouTube video gave a great rundown on how dots and boxes works on paper, explaining the rules well.

Summary of Project

To summarise, I intend to create a virtual game of dots and boxes using a minimax in order to have an AI play the game and act as an opponent to the player. The AI will have the ability to place lines on the playing area, gauging the best move it will be able to make, competing against the player and reacting to the player's moves. The AI will hopefully be able to participate in a turn-based game such as this effectively, adapting its algorithm to different difficulty levels to play slightly differently.

The Current System and the Intended System

Currently, a game of dots and boxes is a paper-based game between 2 human players. The playing area will consist of a grid of 4x4 dots drawn by the player and each player will take turns to place lines on the grid to form boxes. As a box is claimed, the player may label it/colour it with their name/colour and will receive another turn. To recreate this, I will need to implement a similar system visually on a screen and then create an AI to act as player 1's opponent. It will need to be able to efficiently calculate where the best move could be, taking into account all possible moves the human player could make as well as where there are unfinished boxes it could complete to increase its score. There will need to be a turn-based system in place, and each player will need to be distinguishable from each other (i.e. by displaying placed lines and claimed boxes in a colour-coded fashion). A score system will also be required to display the total number of boxes claimed by either player in order to calculate the winner at the end.

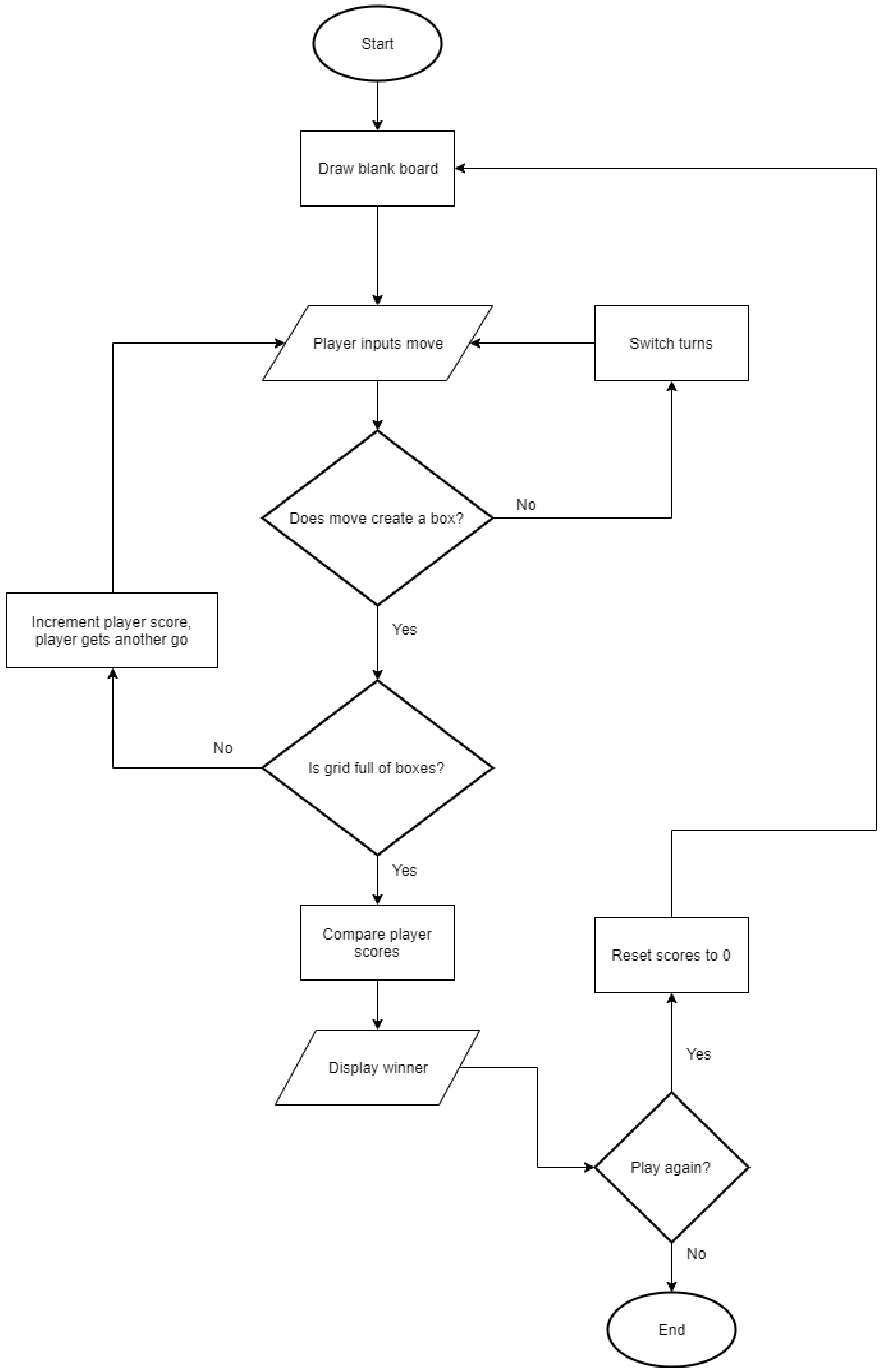
IPSO Diagram

Below is an outline of the potential inputs, processes, storage and outputs the system will need to have:

Input Place line Choose difficulty View AI scores (open CSV) User chooses at the end of the game if they want to continue or not	**Process** Draw game board Switch player turn Determine where to display line Calculate where to move (AI player) Increment box count for either player (score) Calculate if any spaces are left
Storage Total AI wins (CSV file) Internal: - All dot locations - All taken line locations - All available line locations - Boxes made	**Output** Display game board Display lines Display claimed boxes Display current score for either player Display which player's turn it is

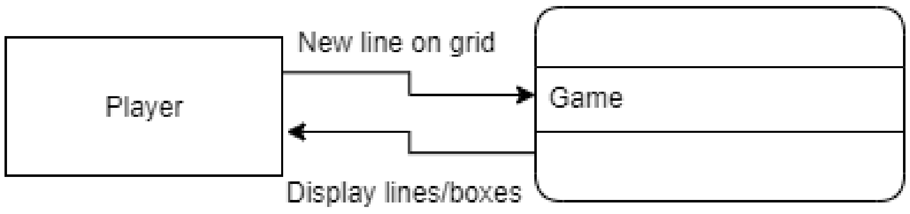
Flow Chart of Current Game Mechanics

Here is a basic flow chart of the game in its current system, with 2 human players:

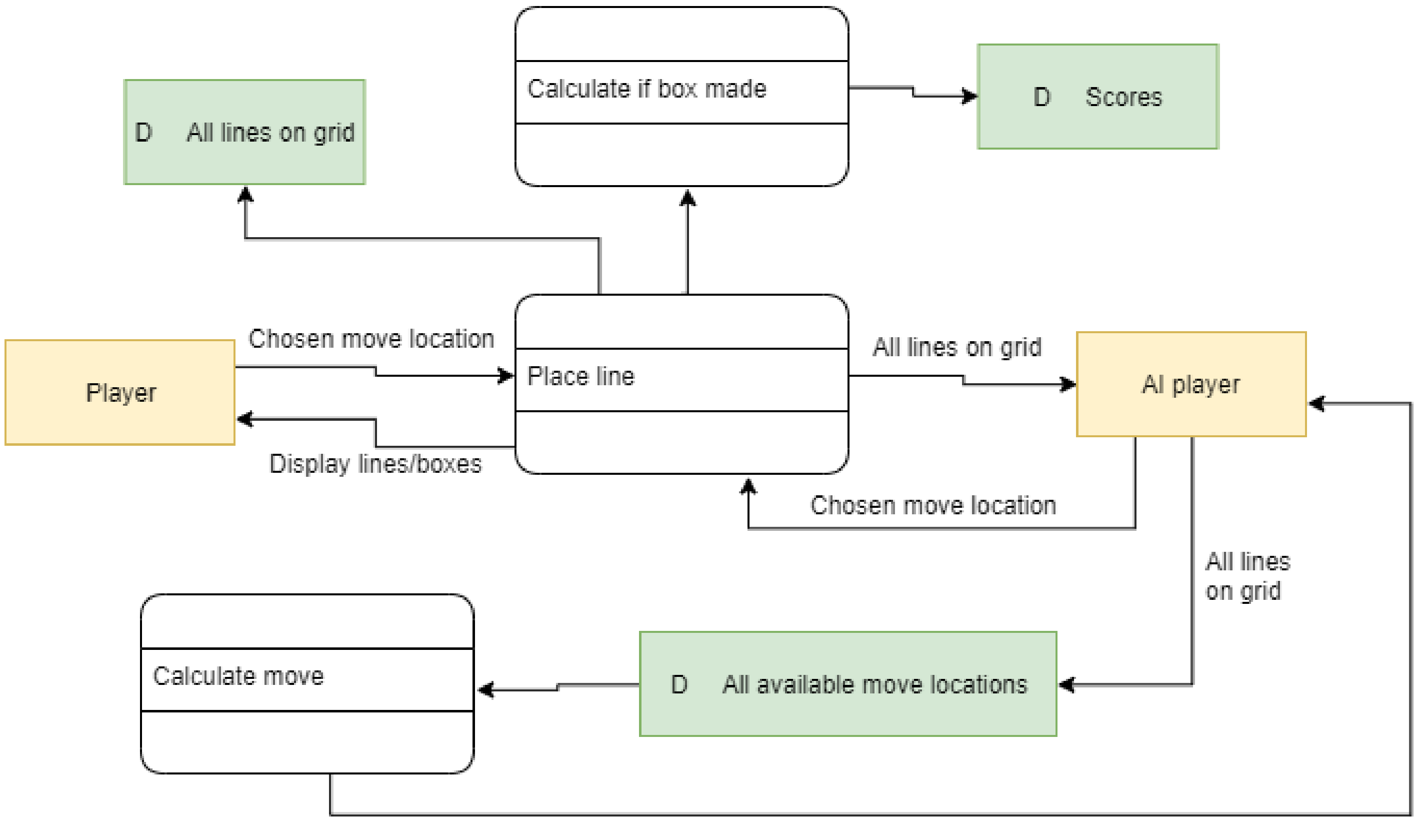


Data Flow Diagrams

Level 0



Level 1

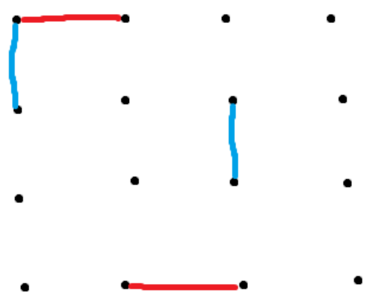


The diagrams above show the overall concept of the mechanics of the game. The player inputs to the game by placing a line on the grid (e.g. entering a coordinate or clicking on the desired line location), which will then be displayed and stored as a used location on the grid. The game will calculate whether a 1x1 box has been formed and if so, add to the appropriate player's score. The AI player will acknowledge all of the current lines on the grid, and calculate where to move from the remaining available spaces, returning its chosen move as a result. The human player will be able to see the game board, as well as where all of the lines and boxes are.

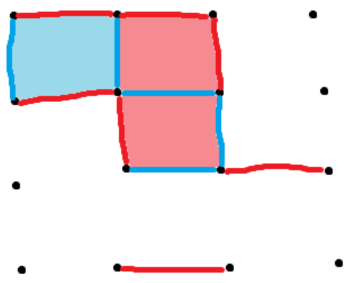
Game Screen and the AI Player

The game screen will need to be visually displayed as something similar to a 4x4 grid of dots where the user can place a line between any 2 dots that have not already been filled by themself or the AI opponent. Both players will require a colour coding system, such as blue for the player and red for the AI, where whenever the user makes a move this is displayed as a red line and for the AI a blue line.

Example:



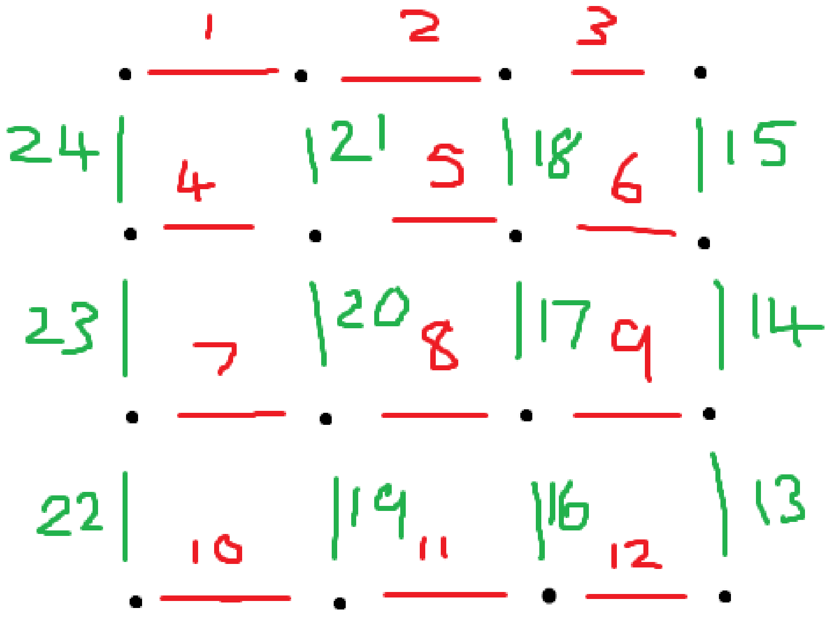
As a box is completed, the region's colour should change to the same or a similar colour to the player who finished the box, earning them a point, like this:



Or alternatively a representation of the player, such as a suitably coloured "P1" or "P2".

Each dot on the screen could be split into a coordinate as part of a list of point values of all given dots. The dots' coordinates relative to the game area could be stored in some kind of list, giving each dot a value within the list.

On a 4x4 grid, there are a total of 24 playable moves either player can make:

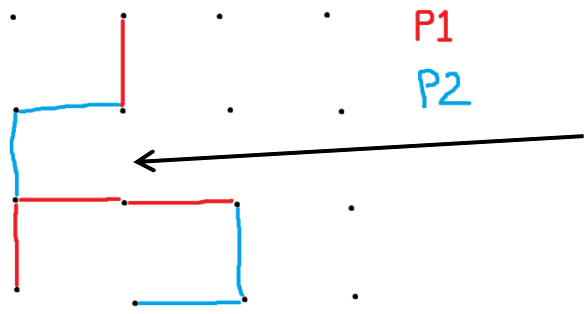


This means that, provided the user player goes first, the AI player will need to take into account the remaining 23 locations that it can go on the first turn. With each new turn for the AI player, generally, this count will decrease by 2 (its own previous turn + player 1's turn) on a normal move – provided the user or itself didn't previously finish a box. The minimax should be able to discern what the best move would be for each turn – starting off fairly random, and as the player makes more inputs, will make a more educated decision. Difficulty will also need to be considered, using less layers of maximising and minimising to create the effect of lower and higher difficulties.

Example of Game Scenario: Continuous Turns

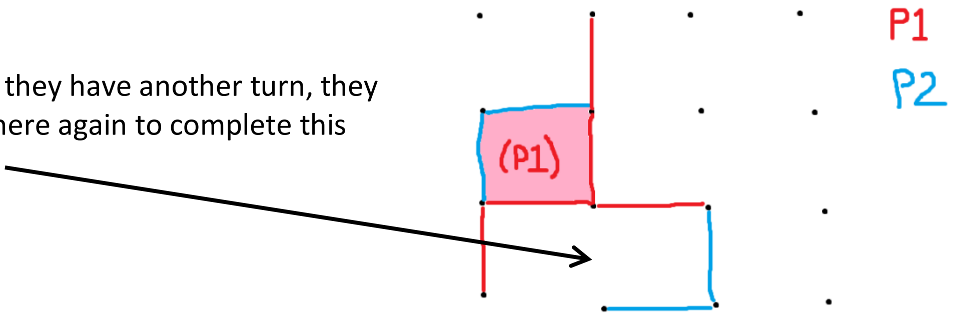
In the instance that a player completes a box in this game, they receive another turn. That player can actually take yet another turn if the first extra turn completed a second box, and this will continue until the player makes a move that does not complete a box.

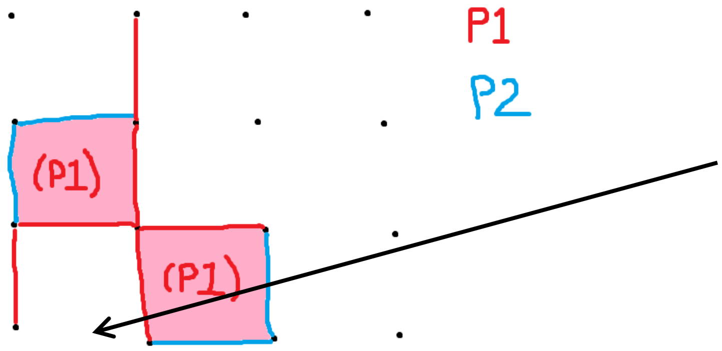
For example:



Here, player 1 is about to take another turn. They can make a move here to complete a box.

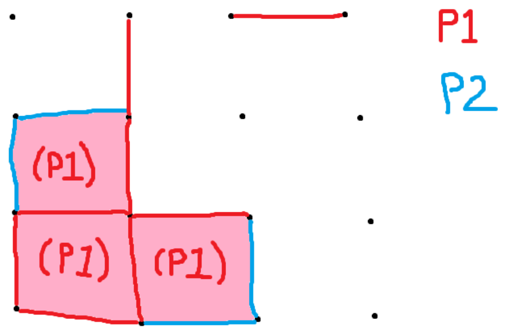
However, since they have another turn, they can also move here again to complete this box.



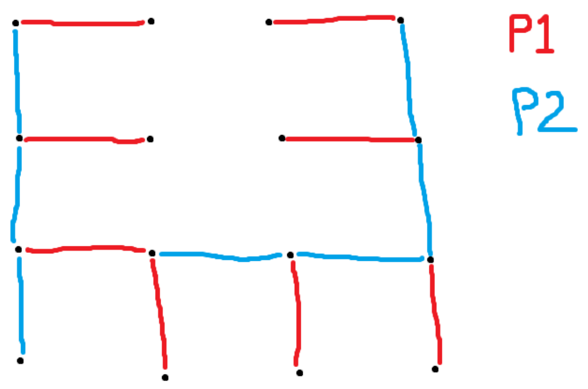


And again, here.

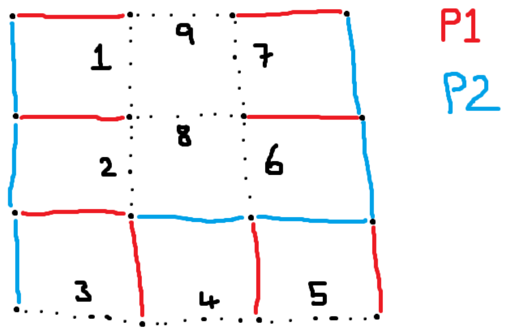
After the final box, player 1 no longer has any boxes it can complete so it is back to a normal turn. The move in this instance is made in the very top right, and at this point it is player 2's turn again. This therefore shows how the "continuous turn" mechanic needs to work.



Theoretically, it would be possible for a single player to acquire all nine boxes in a single go, however this is extremely unlikely and would require the grid to look something like this:



In this particular scenario, seven complete turns have been made, with player 1 placing their eighth. It's player 2's eighth turn at this point, and can claim all nine boxes at once.



In order to alter the minimax algorithm to adjust to different levels of difficulty, the minimax will go through less/more layers of the tree. On an easier difficulty it will iterate through fewer layers so it considers less possible outcomes, while harder difficulties will consider more outcomes of the move they make can have, making it think farther ahead. Even on a hard difficulty, I need to ensure that the minimax algorithm does **not** iterate through every potential outcome in the earlier stages of the game, or else the AI will become the "perfect" player – something which is unrealistic, and most importantly not fun or useful to the user. It defeats the object of the game.

Problem Breakdown

Concluding this analysis, I understand that the system will need a number of elements, including:

• a graphical interface: consisting of a 4 x 4 game board of dots, where the user/AI can place a line between any two dots in its own individual colour

• a request to the user for the difficulty level desired

• a system that will adjust the amount of moves in advance the minimax algorithm considers based upon the chosen difficulty level

• a scoring system: this will track how many boxes each player has in order to identify the winner upon completion of the game

• a function that will recognise when a box has been completed and will colour/mark this box to represent the player who completed it and increment this player's score by a single point

• functions that will ensure both players cannot overwrite each other's lines

• having the players switch turns each time, unless the previous player claimed a box, in which case that player may go again

• a function that will ensure that the game ends when there are no more spaces left to fill on the grid

• use of a minimax algorithm for the AI player that can take all of the available moves, minimise the greatest chance of loss and maximise the loss for the opponent, and be able to make an educated decision each turn that will render it a suitable opponent relative to the difficulty level to a human player

• store total number of games won by AI player for each difficulty level in a text file

Requirements

Overall List of Objectives

1. Create a game board of dots, e.g. 4x4, where the user is allowed to interact with it
    1.1 Dots an even space apart
    1.2 Store each dot as a coordinate on the grid
2. User selects difficulty
    2.1 Selection between easy, medium and hard
    2.2 Adjust number of scenarios minimax algorithm will iterate through based on difficulty selected
3. Have a systematic turn-based system that lets both players have a turn one after the other
    3.1 Check if last move completed a box. If it did, player receives another turn until a normal move is made
    3.2 On a normal move, switch turns after each player moves
4. Enable the user to place lines on the game board and display these lines visually on the board
    4.1 Accept user's click location as location to place line
    4.2 Ensure click is ignored if it does not correlate accurately enough to a space on the grid/overwrites another line
    4.3 Display line on grid in correct colour and relative to where the user clicked
    4.4 Record all line positions on grid
5. Have the scores for both players displayed to the side for the user to see at all times
6. Increment the score if a box is made
    6.1 Display updated score each turn
7. Display the claimed boxes visually
8. Have a functioning AI player that can effectively take into account all the moves both it and the user can make with each turn, making a decision via an implemented minimax algorithm
    8.1 AI iterates through fewer/more depth levels of the minimax depending on the difficulty, to alter how far it can think ahead and thus how "intelligent" it is
    8.2 Record and update all moves available to AI player each turn
9. End the game when the board has been filled
    9.1 Ask if the user wants to replay
10. Calculate and display the overall winner of the game
    10.1 If the AI player won, add this to total games won in another file for corresponding difficulty, having this data accessible within the system. If the AI player lost, only update total games count

Requirements: In Detail

Following the identified overall requirements of my intended system, below are descriptions of how each mechanic will need to work and what it will do for the system.

Inputs

• <u>Choosing where to place a line</u>
The system will firstly need to take the user's input of where they wish to place a line on the grid, for example by taking the position of the mouse click on the window. The user could alternatively enter their desired line location in the form of typed coordinates, however I think that this would be too time-consuming for the user themself and less user-friendly overall. The system will need to ignore any clicks that don't correlate to a proper location, and should not count as a move. The user should click within certain set bounds of an area on the grid, and anything outside should be ignored.

• <u>User selects level of difficulty</u>
This could be between easy, medium and hard, for example. The user will need to select which level of AI they wish to play against, which will in turn modify the minimax algorithm slightly depending on the difficulty chosen.

• Restarting/ending the game
Upon completion of a game, it may be a good idea to have the system ask the user whether they wish to play another round or whether they want to stop. If they want to continue, the user should be able to confirm this and the grid will be redrawn. Otherwise, the game should close. This gives some flexibility to the game, allowing the user to easily replay if they wish to.

Processes

• Constructing the game grid/board
The system will need to draw the 4x4 system of dots on to a Windows Form, where the game will be played. It should place each dot an equal distance apart, taking into account the overall size of the Forms window. I may include additional sizes such as 5x5 and 6x6 for the option of longer games.

• Switching player turns
The system needs to switch between the user player and the AI player with each turn. It should calculate whether the last move either player made was a move that incremented their score (finished a box), and if so, allows that player an additional turn until they make a regular move, in which case the turn will switch to the other player.

• Drawing a line on the grid in the intended location, in the correct colour
Once the user makes a move, the system should be able to display the line in the corresponding location in the user's colour (red). The AI player making the move should have their line drawn and displayed as blue, so the user is able to identify who has gone where on the grid. This is not absolutely necessary, however it makes it more visually easy to look at and analyse and so the user can review each move.

• Calculating if a box has been made and colouring it correctly
If four lines connect and form a 1x1 box, the system will need to increment the score of the player that completed the box (placed the fourth line) and colour the inside of this box with either player's colour (red/blue).

• Calculating if line location is taken
If the user clicks a spot to place a line that either they or the AI player had already used, this should not count as a move and should be ignored until the user makes a valid move where there is a space. The user needs to be prevented from overwriting lines, as this breaks the game.

• Minimax
The minimax will need to be implemented to the AI player and will need to process a number of things. It will firstly need to acknowledge where all of the available spaces are, where the user's and its own lines are, and all of the locations the user can potentially move to, making a decision based on what minimises the maximum amount of loss that is possible. It should know that placing a fourth line on a 3-line box increments its score (its aim), placing a 3rd line decreases the score, placing a 2nd line would mildly increase the score and placing a line in an empty space should be neutral. Depending on the difficulty, the minimax will iterate through fewer or more layers of potential scenarios throughout the game as well. This ensures that the AI will "think" less or farther ahead depending on the difficulty, making it less or more intelligent. Different depth levels will allow for different levels of being able to think ahead – meaning a higher depth will equal a more advanced AI player.

• Determining if the grid is filled
If the grid is completely full of lines and boxes, the system should end the game and calculate the winner, and then ask if the user wants to play again.

Storage

• Current score for both players

The system needs to have a score counter for both players that will store their scores – which will be calculated from the number of boxes they have completed each. This needs to be stored so that it can be displayed for the user to review the current stats for the game, something I think would be helpful and very user-friendly for a game like this.

• The previously calculated winner

At the end of the game, the overall winner of the game should be stored, so that it can then be displayed to the user.

• Used line locations

In order to be able to calculate whether the user has made a valid, unused move as explained earlier, the system needs to store the locations that have already been taken so that they cannot be overwritten and so that the AI also knows that these are no longer valid move locations.

• The number and location of all moves available to the AI

In order to implement the minimax algorithm, the AI will need to know how many locations it will need to take into account in its decision and where all of these are in order for it to operate. It could be stored in a different list of moves and updated each turn, removing moves that have been taken by either player, so the minimax always has access to every move available to it

• Storing total number of AI wins (writing to text file)

In terms of my investigation, this allows me to see how many times the AI wins against the player depending on the difficulty.

Outputs

• The game grid/board

The game board will need to be drawn to the Windows Form for the user to see and interact with.

• All lines (moves)

The game will need to graphically display all of the lines on the grid, distinguished by player moves (red) and AI player moves (blue), where each line is clearly displayed as connecting between 2 specific dots on the grid/board.

• Claimed/completed boxes

The grid should colour the back of the grid where 4 lines intersect as a box in the colour of the player who made the box, or otherwise mark it clearly as P1 or P2. This will help to visually display where either player has scored, staying true to the original design of dots and boxes.

• Whose turn it currently is

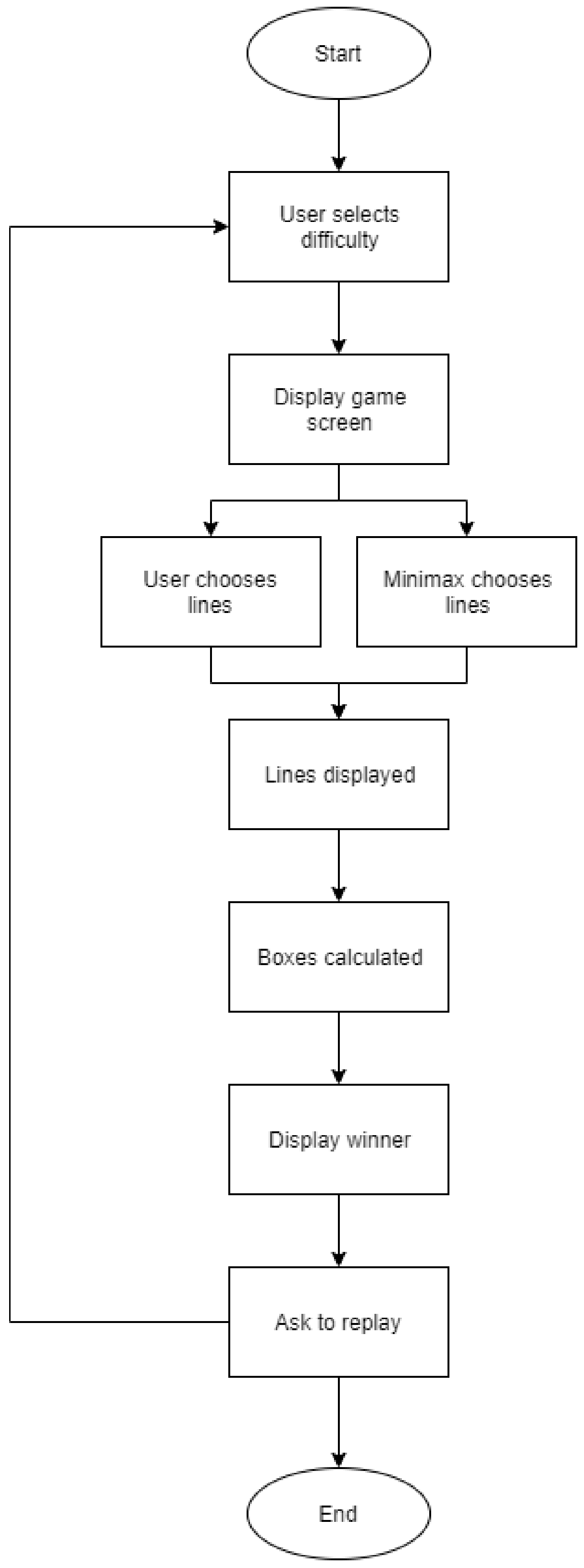
The system could display whose turn it is to make a move, in order to prompt the user player to play when it's their turn, and to just make it overall more visually friendly.

• Current score count for both players

To the side of the grid, there should be a score count for both players that shows how many boxes either player has. This helps the player keep up to date quickly and easily with the stats of the current game

Design

Top Level Design



This basic flow chart describes the main flow of the system.

First, the user will be presented with some form of selection between an easy, medium and hard difficulty, and upon choosing this, they will be directed to the main game screen.

The player and the AI player will then take turns to select lines on the grid, which the system will in turn display back to the user.

Any boxes made get displayed when 4 lines intersect, labelled with the associated player (P1/P2).

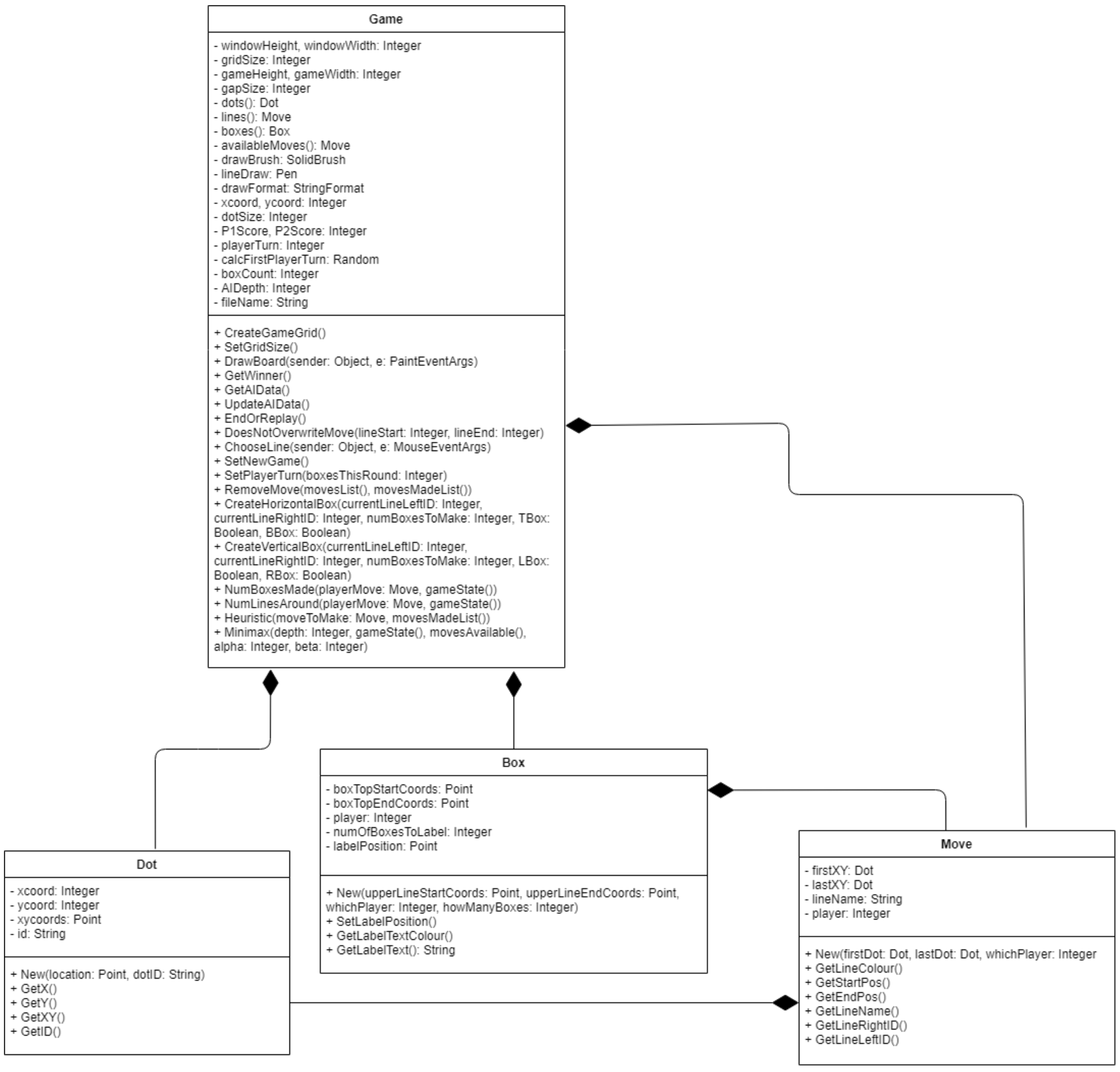
As soon as all boxes have been made (so 9 for a 4x4 grid), the game ends and the winning player is displayed to the user.

They will then be asked whether they wish to replay the game or not. In the event that they want to replay, the process repeats, however if the user selects no the program should close.

Class Diagrams

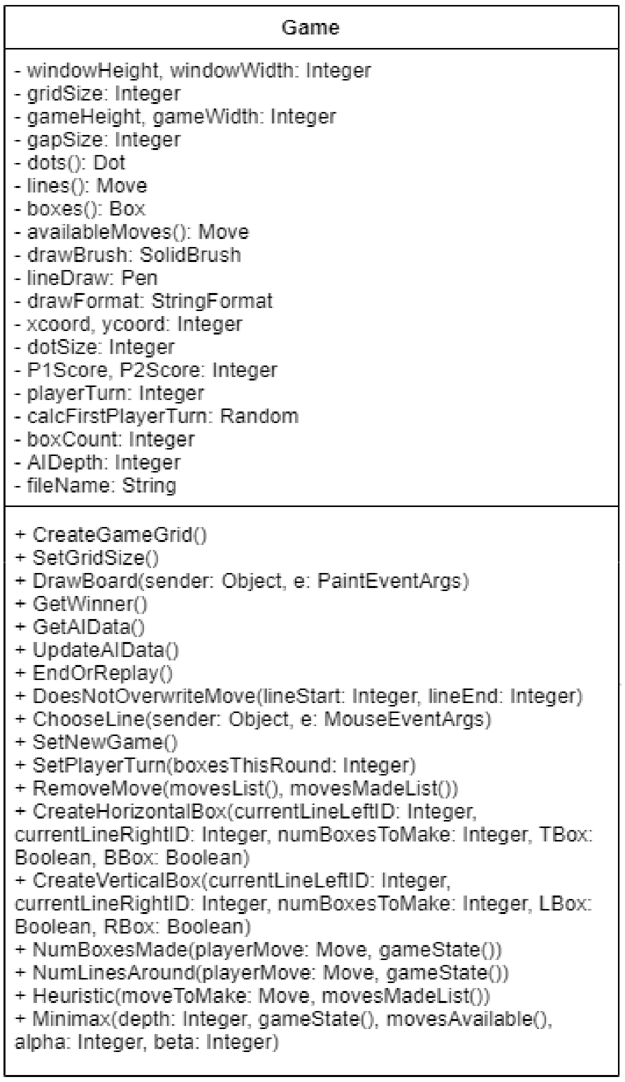
I intend to use an object-oriented model for my system. This will involve the base class, which will feature procedures such as handling the painting of graphics objects (lines, dots, boxes) to the control and handling the player's mouse click. It will also feature classes including a Dot, Move and Box class that handle the dots, lines and boxes being manipulated. I also want to use Windows Forms to create the user interface.

Below are some class diagrams outlining the structure of my main system:


Game
- windowHeight, windowWidth: Integer
- gridSize: Integer
- gameHeight, gameWidth: Integer
- gapSize: Integer
- dots(): Dot
- lines(): Move
- boxes(): Box
- availableMoves(): Move
- drawBrush: SolidBrush
- lineDraw: Pen
- drawFormat: StringFormat
- xcoord, ycoord: Integer
- dotSize: Integer
- P1Score, P2Score: Integer
- playerTurn: Integer
- calcFirstPlayerTurn: Random
- boxCount: Integer
- AIDepth: Integer
- fileName: String
+ CreateGameGrid()
+ SetGridSize()
+ DrawBoard(sender: Object, e: PaintEventArgs)
+ GetWinner()
+ GetAIData()
+ UpdateAIData()
+ EndOrReplay()
+ DoesNotOverwriteMove(lineStart: Integer, lineEnd: Integer)
+ ChooseLine(sender: Object, e: MouseEventArgs)
+ SetNewGame()
+ SetPlayerTurn(boxesThisRound: Integer)
+ RemoveMove(movesList(), movesMadeList())
+ CreateHorizontalBox(currentLineLeftID: Integer, currentLineRightID: Integer, numBoxesToMake: Integer, TBox: Boolean, BBox: Boolean)
+ CreateVerticalBox(currentLineLeftID: Integer, currentLineRightID: Integer, numBoxesToMake: Integer, LBox: Boolean, RBox: Boolean)
+ NumBoxesMade(playerMove: Move, gameState())
+ NumLinesAround(playerMove: Move, gameState())
+ Heuristic(moveToMake: Move, movesMadeList())
+ Minimax(depth: Integer, gameState(), movesAvailable(), alpha: Integer, beta: Integer)
Box
- boxTopStartCoords: Point
- boxTopEndCoords: Point
- player: Integer
- numOfBoxesToLabel: Integer
- labelPosition: Point
+ New(upperLineStartCoords: Point, upperLineEndCoords: Point, whichPlayer: Integer, howManyBoxes: Integer)
+ SetLabelPosition()
+ GetLabelTextColour()
+ GetLabelText(): String
Dot
- xcoord: Integer
- ycoord: Integer
- xycoords: Point
- id: String
+ New(location: Point, dotID: String)
+ GetX()
+ GetY()
+ GetXY()
+ GetID()
Move
- firstXY: Dot
- lastXY: Dot
- lineName: String
- player: Integer
+ New(firstDot: Dot, lastDot: Dot, whichPlayer: Integer
+ GetLineColour()
+ GetStartPos()
+ GetEndPos()
+ GetLineName()
+ GetLineRightID()
+ GetLineLeftID()

Main Game Class

The main Game class will serve as the base class. It will contain many values that other subs within the class will interact with.



```
Game
- windowHeight, windowWidth: Integer
- gridSize: Integer
- gameHeight, gameWidth: Integer
- gapSize: Integer
- dots(): Dot
- lines(): Move
- boxes(): Box
- availableMoves(): Move
- drawBrush: SolidBrush
- lineDraw: Pen
- drawFormat: StringFormat
- xcoord, ycoord: Integer
- dotSize: Integer
- P1Score, P2Score: Integer
- playerTurn: Integer
- calcFirstPlayerTurn: Random
- boxCount: Integer
- AIDepth: Integer
- fileName: String

+ CreateGameGrid()
+ SetGridSize()
+ DrawBoard(sender: Object, e: PaintEventArgs)
+ GetWinner()
+ GetAIData()
+ UpdateAIData()
+ EndOrReplay()
+ DoesNotOverwriteMove(lineStart: Integer, lineEnd: Integer)
+ ChooseLine(sender: Object, e: MouseEventArgs)
+ SetNewGame()
+ SetPlayerTurn(boxesThisRound: Integer)
+ RemoveMove(movesList(), movesMadeList())
+ CreateHorizontalBox(currentLineLeftID: Integer,
currentLineRightID: Integer, numBoxesToMake: Integer, TBox:
Boolean, BBox: Boolean)
+ CreateVerticalBox(currentLineLeftID: Integer,
currentLineRightID: Integer, numBoxesToMake: Integer, LBox:
Boolean, RBox: Boolean)
+ NumBoxesMade(playerMove: Move, gameState())
+ NumLinesAround(playerMove: Move, gameState())
+ Heuristic(moveToMake: Move, movesMadeList())
+ Minimax(depth: Integer, gameState(), movesAvailable(),
alpha: Integer, beta: Integer)
```

Declarations

- **windowHeight, windowWidth**: sets window height and width in pixels

- **gridSize/gapSize**: sets size of grid, which will be 4 for a 4x4 grid for example, while gapSize sets size of gap between dots

- **gameHeight, gameWidth**: this will handle the pixel dimensions of the panel the main game will take place on

- **dots()/lines()/boxes()**: lists that will hold the locations and details of each dot on the grid, each placed line and created box

- **drawBrush/lineDraw/drawFormat**: variables required for the DrawBoard sub later that handles all of the paint events on the game space

- **dotSize**: constant integer value that will determine pixel size of each dot

- **P1Score, P2Score**: integer values that will hold the values of both players' scores

- **playerTurn**: integer value that will hold either a 1 or a 2, indicating which player's turn it is

- **AIDepth**: integer that will hold number of depth levels AI needs to iterate through based on user choice

- **availableMoves()**: all moves available for the AI player to use

- **boxCount**: holds current number of boxes on the grid

- **fileName**: string that will store the file name of the CSV I intend to use to store score data

Functions and Procedures

- **CreateGameGrid**: will contain a double for loop (through x then y) that iterates from 0 through to the gridSize subtract 1 (0 to 3, which will create all 16 dots, correlating to each dot index), adding each dot to the dots list to be drawn to the panel

- **SetGridSize**: sets the size of the grid as selected by the user (4x4/5x5/6x6)

- **DrawBoard**: procedure that handles the pain events. This involves processes such as drawing all the dots, all of the lines and all of the boxes by iterating through their associated lists

- **ChooseLine**: procedure that handles the mouse click and overall operation of selecting a line. It features getting mouse coordinates and translating them into lines, and procedures for checking boxes are handled here

- **SetNewGame**: a procedure that resets values like scores, moves, the box count etc. to their default settings of 0 in order to set a fresh game

- **CreateHorizontalBox/CreateVerticalBox**: 2 procedures that handle identifying where a specific box has been claimed, handling the labelling of boxes

- **GetWinner**: determines and returns the winning player at the end of a game

- **NumBoxesMade**: a function that will return an integer value on each turn to determine how many, if any, boxes have been made. This is used to increase the score and also as part of scoring for the heuristic function

- **SetAIDepth/GetAIData/UpdateAIData**: subs for the AI that will set the AI depth to variable AIDepth based on user's difficulty choice, retrieve current AI score data from the CSV and subsequently update them

- **EndOrReplay**: sub that asks the user if they wish to replay or quit

- **DoesNotOverwriteMove**: sub that will check if the location the user clicks will overwrite an existing line

- **SetPlayerTurn**: sub that determines if the player turn needs to be switched and does so if necessary, and update the player score/turn labels

- **NumLinesAround**: to be used in the minimax heuristic: calculates number of lines around given move to calculate its heuristic score and how effective a move it would be to the AI

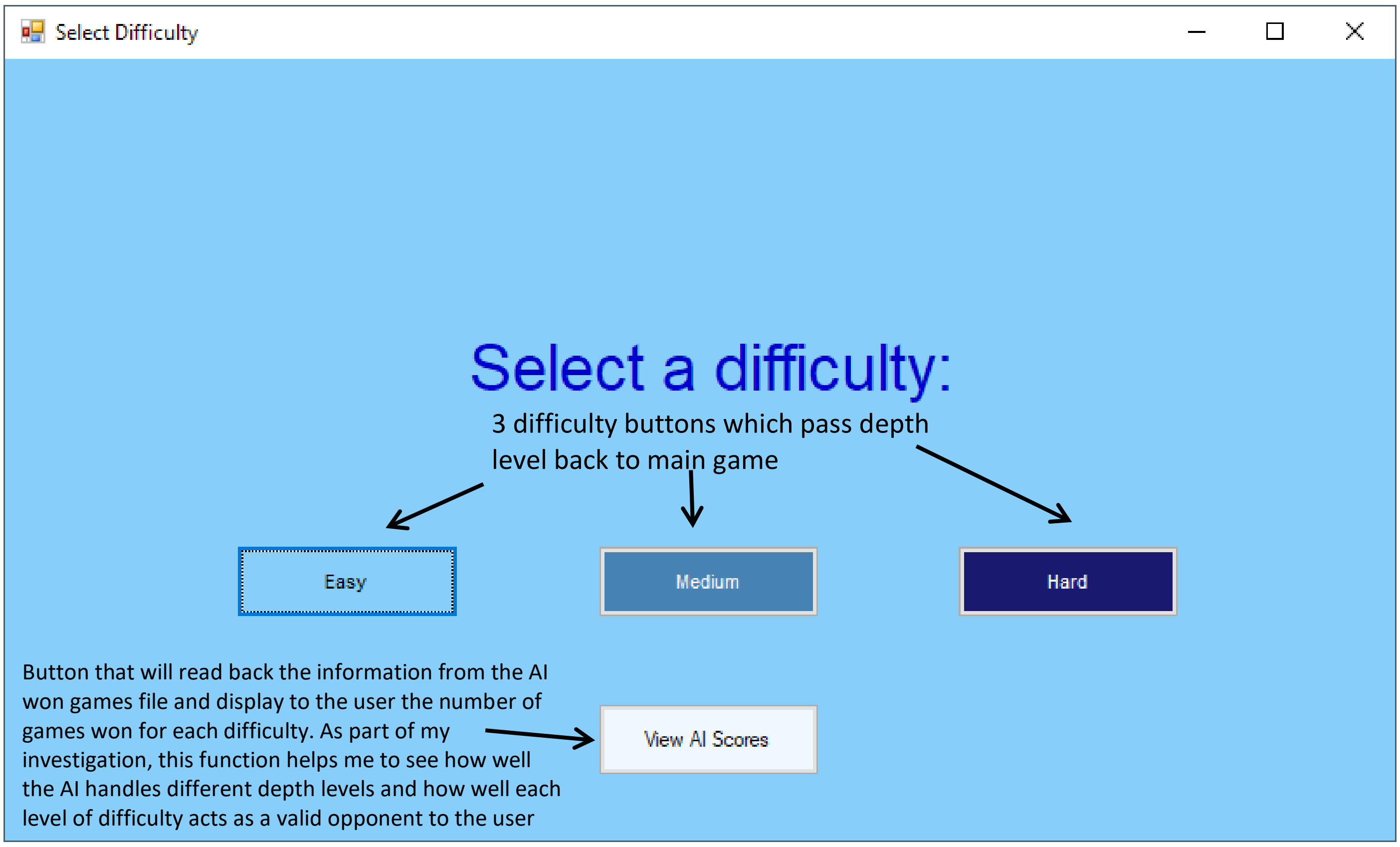
- **Heuristic /Minimax**: subs that will score potential moves based on how effective they would be for the AI, returning moves with the greatest score for the AI player, recursively iterating through all of the different game scenarios to judge and return the AI's best possible move

- **RemoveMove**: sub that removes a move from a list of available moves to the AI once it has been used by either player. This can function on the physical game grid for the AI to know its selection of moves available, but also within the minimax within the temporary game states

- **GetAIData/UpdateAIData**: subs that will retrieve the AI score data from the CSV file, update the scores from a completed game within the program and rewrite the data back to the CSV file using the file location stored in the string fileName

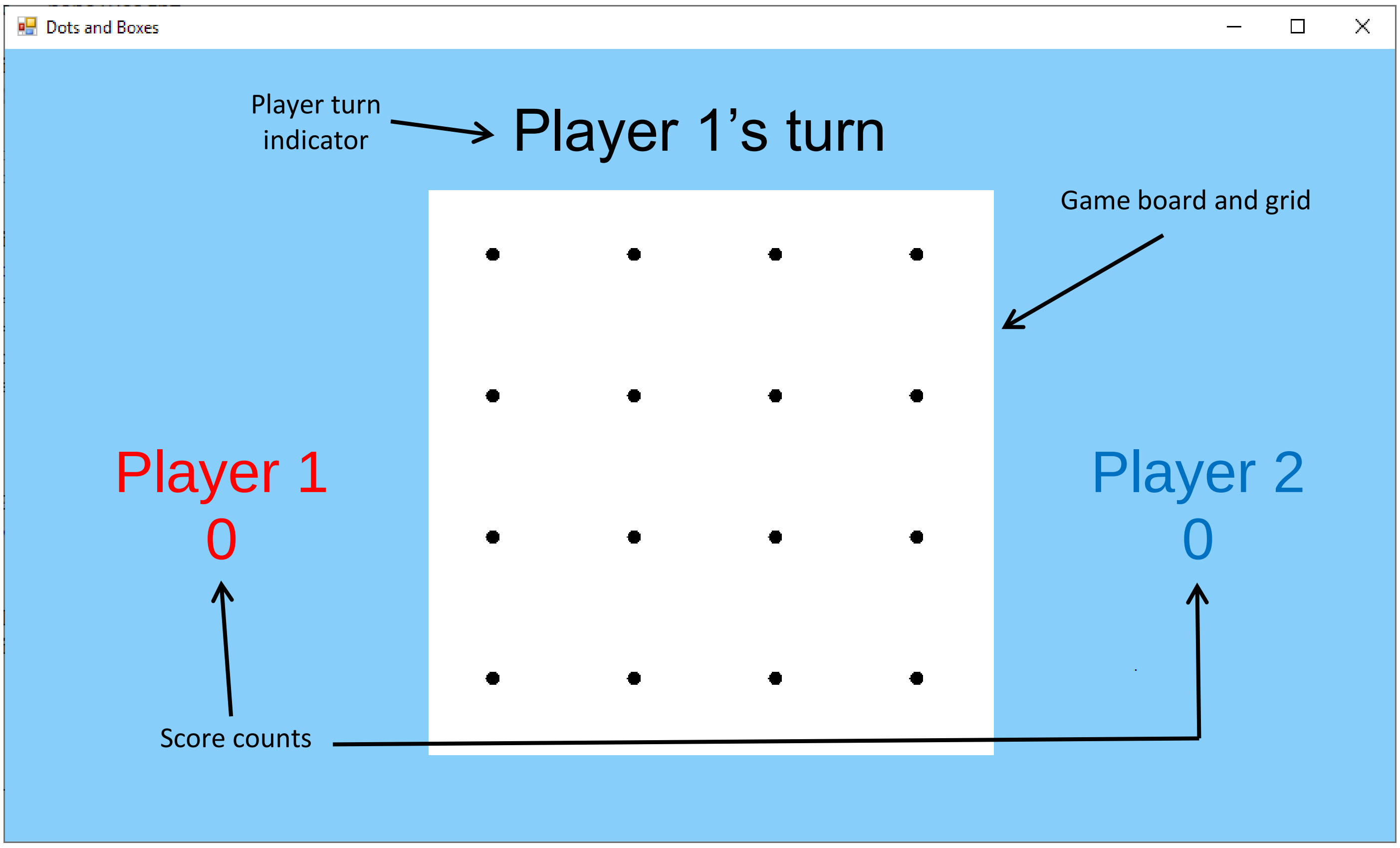
Difficulty Select Screen

Upon opening the program, the user should be greeted with a window similar to the one below that allows them to select a difficulty between easy, medium and hard, and additionally an option to view all the AI win scores for each difficulty from the external file. Depending on the difficulty chosen, this will pass the number of depth levels back to the main game form. If the View AI Scores button is selected, a message box should show, displaying details of the total number of wins for each difficulty, by reading back from the AI won games file.



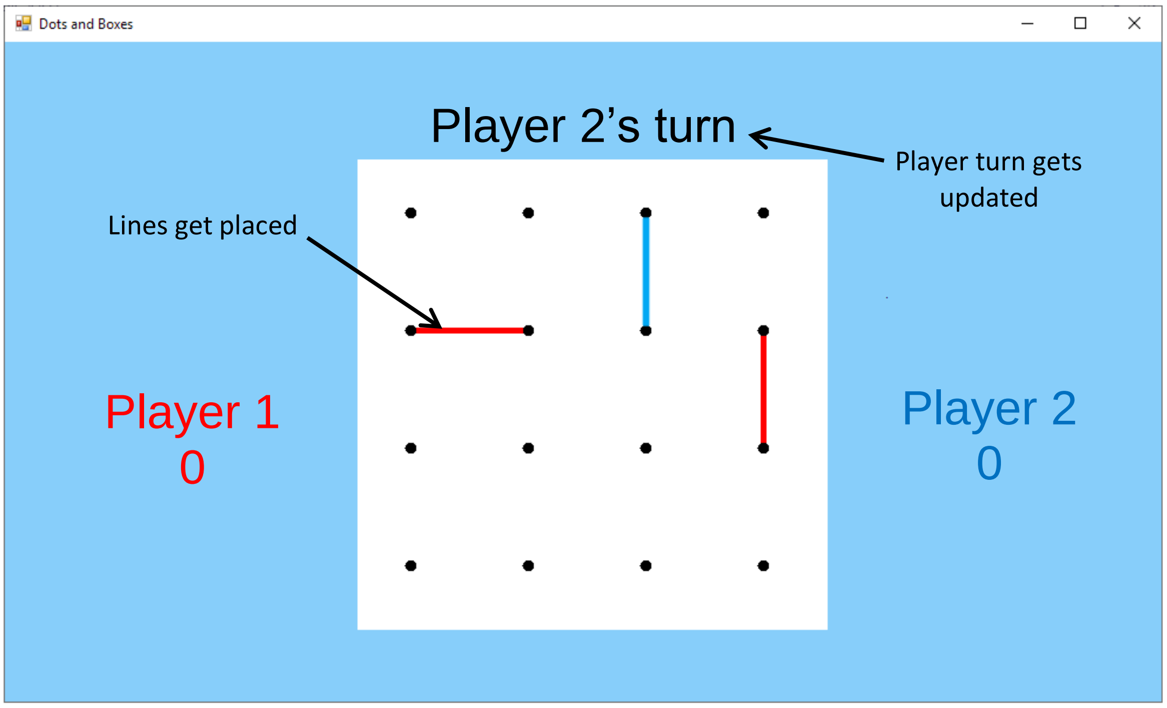
Button that will read back the information from the AI won games file and display to the user the number of games won for each difficulty. As part of my investigation, this function helps me to see how well the AI handles different depth levels and how well each level of difficulty acts as a valid opponent to the user

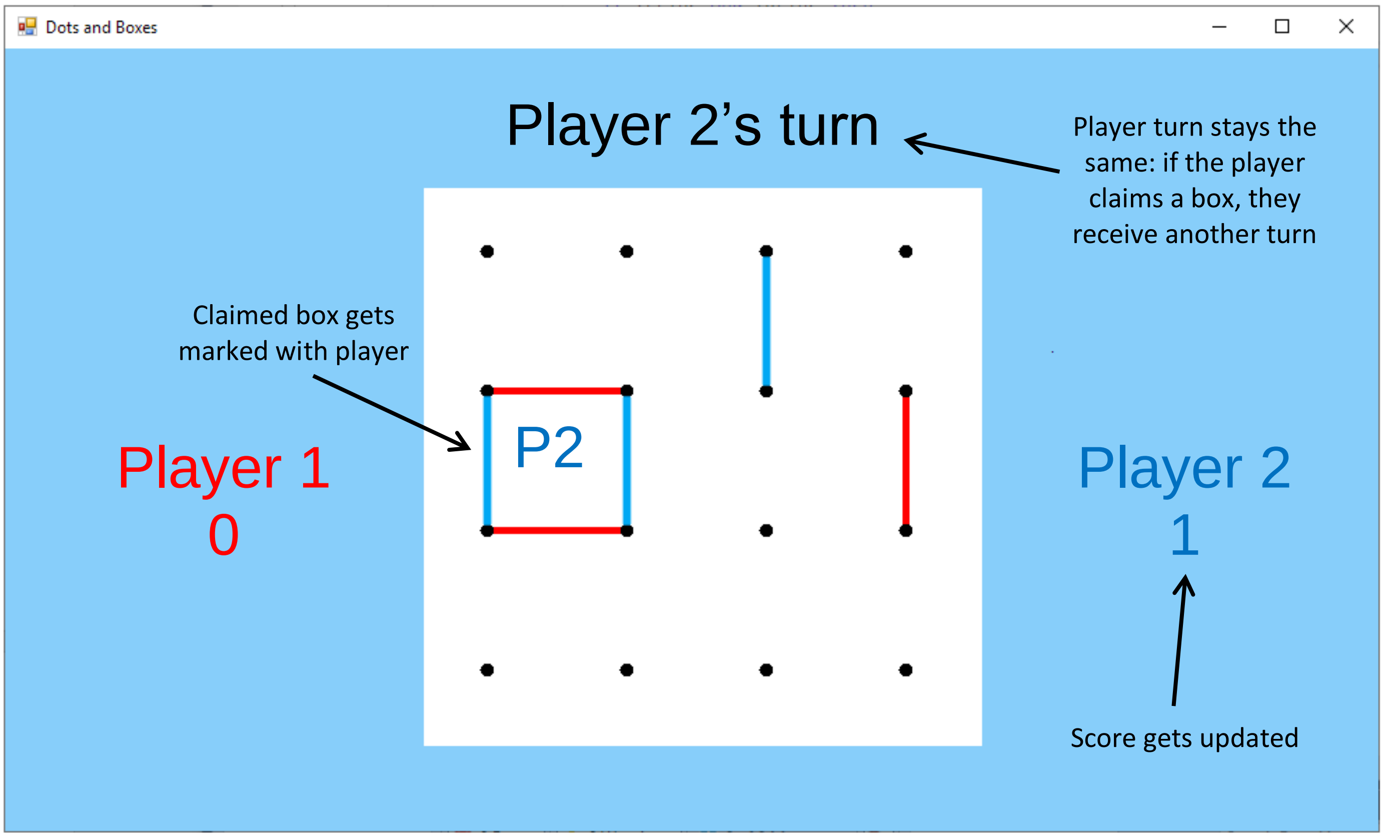
Game Screen



Next, after choosing a difficulty level, the user should be greeted with a screen similar to the one above. It should feature a panel in the centre, composed of an evenly spaced 4x4 grid of dots. Either side you should find the score count for both player 1 and player 2, in their corresponding colours. This count will update each turn. The top should specify whether it's player 1's turn or player 2's turn.

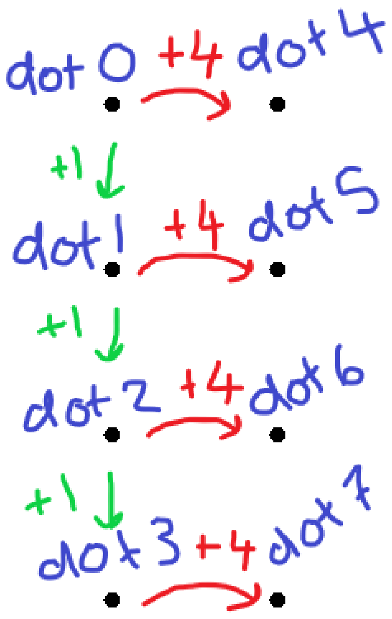
As the game progresses, lines should get displayed on to the screen, and if four lines intersect, a box should be displayed and marked with the player who earned it, and the score of that player increases by 1.



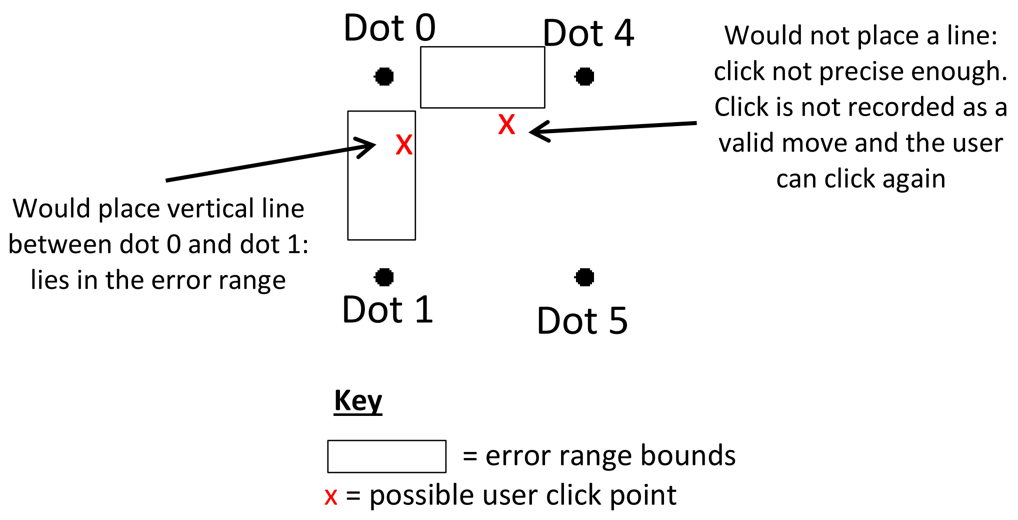

Dots and Boxes
Player 2's turn
Player turn stays the same: if the player claims a box, they receive another turn
Claimed box gets marked with player
P2
Player 1
0
Player 2
1
Score gets updated

Placing Lines

Placing lines for the user should involve them clicking the area in which they want to place the line, which the system should then display. In order to do this effectively, the system needs to take the X and Y coordinates of the mouse click, and compare these against the positions of each dot in the list. If the mouse coordinates fall between a dot and the next dot in the list, it should place the line vertically, and if the dot falls between a dot and the dot 4 index values ahead, it should place it horizontally. The line coordinates are then added to a list of moves, similar to the dots.



I also want the system to have an error range of a certain number of pixels, such as 10, for the user to click within. This means that if they click slightly to the left or right, the system will still accept the click as a valid line placement.


Dot 0
Dot 4
Dot 1
Dot 5
Would not place a line: click not precise enough. Click is not recorded as a valid move and the user can click again
Would place vertical line between dot 0 and dot 1: lies in the error range
Key
= error range bounds
x = possible user click point

Additionally, no line should be placed if the player selects a taken location.

Calculating Lines Pseudo Code

Here is the pseudo code I have designed for the above line placing procedure:

```
FOR dot = 0 TO 15
      TRY
            IF (mouseCoords(Y) > dots(dot)(GetY) + errorRange AND mouseCoords(Y)
            < dots(dot + 1)(GetY) - errorRange) AND ((mouseCoords(X) <=
            dots(dot)(GetX) AND mouseCoords(X) > dots(dot)(GetX) - errorRange)
            OR (mouseCoords(X) >= dots(dot)(GetX) AND mouseCoords(X) <
            dots(dot)(GetX) + errorRange)) THEN
                  vertical = True

            ELSE IF (mouseCoords(X) > dots(dot)(GetX) + errorRange AND
            mouseCOords(X)  < dots(dot + 4)(GetX) - errorRange) AND
            ((mouseCoords(Y) <=  dots(dot)(GetY) AND mouseCoords(Y) >
            dots(dot)(GetY) - errorRange) OR (mouseCoords(Y) >= dots(dot)(GetY)
            AND mouseCoords(Y) < dots(dot)(GetY) + errorRange)) THEN
                  horizontal = True

      END TRY

[...]
```

What the beginning of this for loop does is calculate where the mouse click correlates to, determining whether the line should be vertical or horizontal. If the mouse Y coordinate is larger than the dot and smaller the dot at the very next index, and within the bounderies of the error range, the line is registered as being vertical and will set the next dot index to +1. Similarly, if mouse X coordinate is larger than the dot and smaller than the dot 4 indexes away (on a 4x4 grid) and within the boundaries of the error range, the line is registered as being horizontal and will set the next dot index as +4. The system can then later draw this on to the form as the lines are added to the Move list.

Validating Lines

In order to ensure that the user has clicked a location that corresponds to a location on the grid, I feel that, as in the pseudo code above, a try-catch statement would be appropriate. If there is an exception (e.g. not an appropriate click), neither the vertical nor horizontal boolean will be set to true and the system will make no further attempt to switch the player or advance the code, waiting until an appropriate click is made. Additionally, to make sure that the player cannot overwrite one of their own lines or the lines of the AI player there needs to be more validation. I intend to use a boolean, validMove, which will return false if, after iterating through each move in the lines list, it finds a line there. For example, in the instance of the line being vertical:

```
IF vertical THEN
      FOR line = 0 TO Length (lines) - 1
            IF lines(line)(GetStartPosition) = dots(dot)(GetXY) AND
            lines(line)(GetEndPosition) = dots(dot + 1)(GetXY) THEN
                  validMove = False

      NEXT line

[...]
```

This will check each line in the total list of all lines on the grid against the current mouse click to see if it already exists. If it does, validMove is set to false.

Move Class

As a line is placed on the grid, it will additionally be added to a list of lines stored as object Move, holding a point value for the start position of the line and a point value for the end position of the line, correlating to the coordinates of the 2 dots it connects. It should also hold which player made the move, so that when it comes to drawing the line, the system will know whether to draw it in red or blue.

```
FirstCoords = Point
LastCoords = Point
LineName = String
Player = Integer

New (FirstDot: Dot, LastDot: Dot, WhichPlayer: Integer)
      FirstCoords = FirstDot (Get XY)
      LastCoords = LastDot (Get XY)
      LineName = First Dot (GetID) + "-" + LastDot (GetID)
      Player = WhichPlayer

Function GetStartPos ()
      return FirstCoords

Function GetEndPos ()
      return LastCoords

Function GetLineName ()
      return LineName

Function GetLineName1 ()
      return first part of LineName

Function GetLineName2 ()
      return second part of LineName

Function GetLineColour ()
      IF Player = 1 THEN
            GetLineColour = red
      ELSEIF Player = 2 THEN
            GetLineColour = blue
```

The class will take the first and last coordinates it needs, and will also create a line name based on the ID values of the two dots it draws between (see Dot Class pseudo code). For example, a line being drawn between dots of ID 1 and 5 would give the line the name “1-5”, which I later want to use to check for boxes. It also takes the first and last part of the line name separately so it can check against other values later. If the line name was 1-5, GetLineName1 would return 1 while GetLineName2 would return 5. The GetLineColour function decides whether the line should be coloured blue or red depending on the player’s turn, to be handled within the procedure that handles painting to the control later.

Ending the Game

The game will feature a variable, boxCount, that will increment as boxes are made by either player. Once the value reaches 9 on a 4x4 grid, the game will end, the winner will be displayed and the player is asked if they wish to replay.

Pseudo Code: Setting Up the Dots

The following pseudo code is intended to create the dots by calculating their appropriate coordinates in relation to the game space and adding them to a list of 16 dots:

```
WindowHeight = 600
WindowWidth = 1000
GridSize = 4
DotSize = 10
dots = list of dot
BoardLocation = Point ((WindowWidth - BoardWidth) / 2, (WindowHeight -
BoardHeight) / 2)
DotGap = (1 / GridSize) * BoardHeight
FOR x = 0 TO GridSize - 1:
      FOR y = 0 TO GridSize - 1:
            DotXCoord = (BoardWidth / DotSize) + (DotGapSize * x)
            DotYCoord = (BoardHeight / DotSize) + (DotGapSize * y)
            dots <-- add new dot (DotXCoord, DotYCoord)

      NEXT y
NEXT x
```

This sets the game space in the centre of the game window by taking into account the window height and width, and then sets the gap required between dots in relation to the size of the game board. The GridSize relates to how many dots there are widthways and lengthways, which in a normal game of dots and boxes would be 4. The program then iterates widthways and lengthways through each coordinate on the game board and evenly assigns each dot an X and Y coordinate, then adding these as a dot in the list dots.

Dot Class

In order for dots to be added to a list as above, I want them to be stored as a separate object, Dot. The class will have an X coordinate and a Y coordinate, for each dot, as well as an ID for the dot. In the instance of a new dot being created, such as when dots are being added to the list, this X and Y coordinate and the ID will be taken, and the class will feature functions to return these values for whenever they are required throughout the program.

```
X = Integer
Y = Integer
XY = Point
id = String

New (location: Point, DotID: String)
      X = X of location
      Y = Y of location
      XY = location
      id = DotID

Function GetX ()
      return X

Function GetY ()
      return Y

Function GetXY ()
      return XY

Function GetID ()
      return id
```

draw it to the game space that way, using the SolidBrush object from the VB.net PaintEventArgs class – which handles painting graphics to the control.

```
DotSize = 10
DrawDot = SolidBrush (colour: black)

FOR each dot in dots:
     draw circle to GameBoard (DrawDot, dot (GetX), dot (GetY),
     DotSize)
NEXT dot
```

Identifying Boxes

When any 4 lines intersect, a box needs to be made. This could be done by separating the code into checking firstly for horizontal boxes (up *and* down), and then vertical boxes (left *and* right), done within the line choosing sub that handles the mouse click event. For a horizontal move, for example, the horizontal boolean would be set to true, and the following procedure would apply:

```
IF horizontal THEN
     IF line left ID MOD 4 <> 0 THEN
          FOR each line IN lines
               Check left, right and bottom lines
          NEXT line


     IF left line, right line AND bottom line THEN
          numBoxesToMakenumBoxesToMakeHorizontally += 1
          lowerBox = True


     IF line left ID MOD 4 <> 1 THEN
          FOR each line IN lines
               Check left, right and top lines
          NEXT line


     IF left line, right line AND bottom line THEN
          numBoxesToMakeHorizontally += 1
          upperBox = True


     SELECT CASE numBoxesToMakeHorizontally
          Case 1:
               boxCount += 1
          Case 2:
               boxCount += 2

     IF isBoxMade THEN
          create box (line left ID, line right ID,
          numBoxesToMakeHorizontally, upperBox, lowerBox)
```

If the left line ID (left dot ID) mod 4 is not equal to 0, then the system will check all lines right, left and below the player's move. The reason for this is that start dots 4, 8, 12 and 16 are all of the dots on the bottom row of a 4x4 grid, and, all being multiples of 4, mod 4 would produce 0 for every start dot on this row. This means that the system will ignore checking for boxes below them. Similarly, if the left line ID mod 4 is not 1, the system will again check lines right and left of the player's current moves, but instead of the line below, it will check the line above. This means the system will now include start dots 4, 8, 12 and 16 along the bottom in order to check for boxes above, but will ignore lines with start dots 1, 5, 9 and 13, as these values all produce 1 when mod 4 is applied. This is because these are the top line of dots, so no boxes need to be checked above them. Then boolean values upperBox and lowerBox will be set, indicating which box type is needed for labelling. Additionally, both can be true at the same time, since you can claim 2 boxes simultaneously – hence why the select case statement will increment boxCount by either 1 or 2 depending on how many boxes were claimed. The function isBoxMade will return either true or false depending on whether one or more boxes have been made for that turn, which can be used as a win condition for the AI player.

Vertically, this process is similar. Instead of using mod 4, however, we can just check for line left IDs greater than 4 for boxes to the left, and line left IDs smaller than 13 for boxes to the right, as dots move from top to bottom in step in terms of their IDs.

Switching player turns will be a case of changing playerTurn from 1 to 2/2 to 1 if numBoxesToMakeHorizontally and numBoxesToMakeVertically are both 0. This ensures the players switch turns if they haven't made a box. If either one is 1 or more, this means 1 or more boxes have been made and that player needs to have another turn.

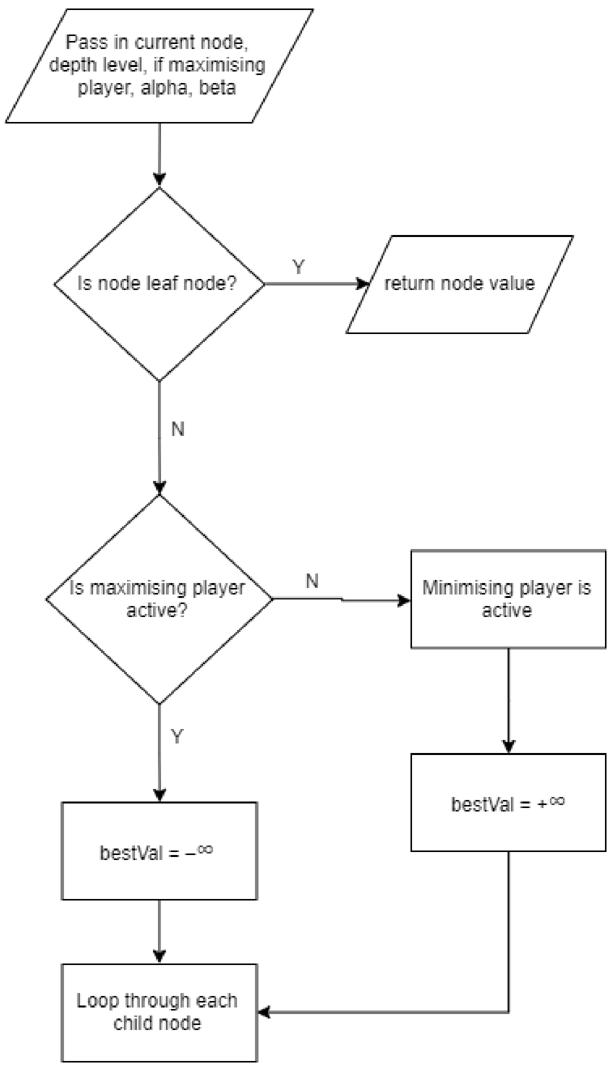
Box Class

The Box class will essentially handle the marking of boxes on the game grid, taking into account the upper line of the box and the player turn in order to know where to place the box label and which player it needs to represent.

```
boxTopStartCoords = Point
boxTopEndCoords = Point
player = Integer
numOfBoxesToLabel = Integer
labelCoords = Point

New (ULineStartCoords: Point, ULineEndCoords: Point, whoseTurn: Integer, howManyBoxes: Integer)
       boxTopStartCoords = ULineStartCoords
       boxTopEndCoords = ULineEndCoords
       player = whoseTurn
       numOfBoxesToLabel = howManyBoxes

Function SetLabelPosition()
       X of labelCoords = (X of boxTopStartCoords + X of boxTopEndCoords) / 2
       Y of labelCoords = Y of boxTopStartCoords + (X of labelCoords + X of boxTopStartCoords)
       return labelCoords

Function GetLabelTextColour()
       IF player = 1 THEN
              GetLabelTextColour = red
       ELSEIF player = 2 THEN
              GetLabelTextColour = blue

Function GetLabelText()
       IF player = 1 THEN
              GetLabelText = "P1"
       ELSEIF player = 2 THEN
              GetLabelText = "P2"
```

Minimax

The AI will need to be able to identify the best move within n depth levels it can make using the minimax. It could use alpha-beta pruning in order to reduce any unnecessary computation time, such as for example checking for minimum values on a particular depth level when a minimum value has already been found. This works by adding additional parameters, alpha and beta, with alpha representing the best value the maximiser can guarantee for a given depth level, and beta representing the best value the minimiser can guarantee for a given depth level. The first time the minimax function is called, alpha would be set to a value equivalent to negative infinity and beta to positive infinity. By comparing each child node with not just the best value but also against alpha and beta, branches in the minimax that don't need to be searched can be ignored this way. The minimax, at the beginning, would generally operate something like this:


Pass in current node, depth level, if maximising player, alpha, beta
Is node leaf node?
Y
return node value
N
Is maximising player active?
N
Minimising player is active
Y
bestVal = –∞
bestVal = +∞
Loop through each child node

If the current node is a leaf node (also known as a terminal node – in this instance, the depth equalling 0 or no further moves being available due to the grid being complete would be a terminal state for the minimax algorithm), this node value is returned as a result of calling the function, therefore passing the AI's chosen move into the game. For each possible move path, these values are compared to find the greatest, and thus the best move.

The minimax begins by maximising. The best value is set to –∞ and the minimax will iterate through every child node (every game state). Within the minimiser the best value is set to +∞, and it then places another move and iterates through the child node of this move, and the minimax continues to recursively alternate between minimising and maximising until we reach the leaf nodes (maximum depth). At this point, the raw score, or heuristic value, of each of these leaf nodes of the move above are checked and provided we are maximising at this point, the leaf node with the greatest heuristic value is passed up to the maximiser (by comparing if each score > best value). Of all the moves at this depth which collect the greatest leaf node value, the minimiser then selects between them the one that would return the lowest score (by comparing if each score < best value) - so we can say here that the role of the minimiser is minimising the maximum amount of success. Once the values have been accumulated for each move on this minimising layer, the maximiser above it gets the greatest value from them. The role of the maximiser here is to therefore maximise the minimum amount of success as calculated in the minimising layer below it. We can therefore see how the minimax algorithm would operate.

The AI will be able to identify the terminal game state by checking if there are no further moves available to make – indicating that the grid is full – using the list containing all of the possible moves available to it.

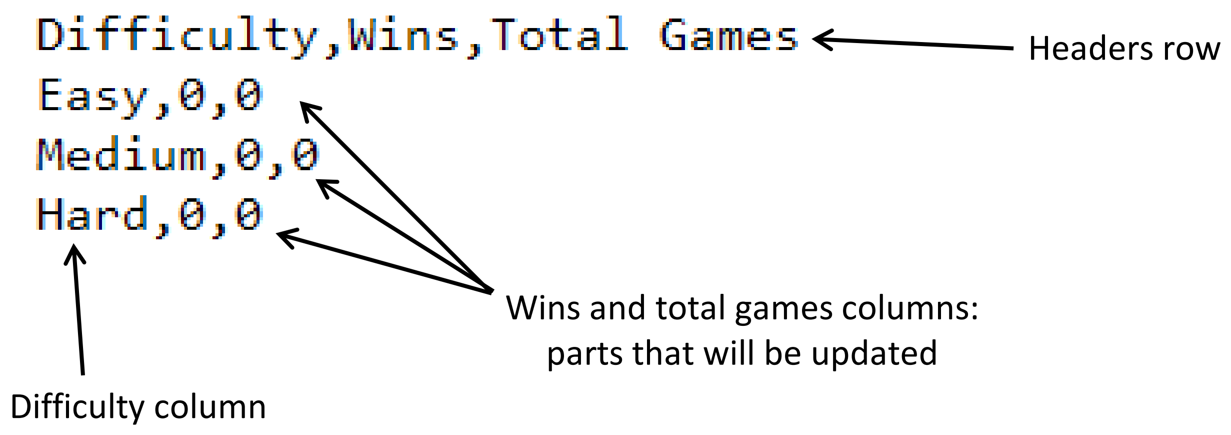
The role of alpha and beta is to reduce computation time. For example, if we were on a maximising layer, getting the greatest possible value and coming from a minimising layer above, a value for beta would have been passed in. Any time a greatest value is set at this maximising layer, alpha changes to it. If at any point beta becomes smaller than or equal to the value of alpha, this means that it is not worth continuing to search down this move branch. The reason for this is that if for example the value of beta from the minimising layer above was -2 (smallest and therefore best value so far at the minimising layer) and alpha at this layer was currently 5 (best value so far at this maximising layer), it is not worth continuing to search this path as the minimiser above is already guaranteed a value of -2 or lower.

Writing AI Results to a Text File

I want the system to be able to count the number of times the AI has won against the player for each difficulty level and store this externally for me to access as part of my investigation. This will enable me to see how efficiently the AI can pose as an opponent for the three different difficulty levels I want to implement to it. If the AI wins a game, it should add 1 to the count in a text file by writing to it in accordance with the difficulty level the user played on. I could use a CSV file to do this. This information can then be used on the main difficulty select screen, where if the user wishes to see the won AI games, they can do so via the use of this file.

```
Structure AIWins
      totalWins = Integer
      difficulty = String
      totalGames = Integer

winner = 2
Open file "aimoves.csv"
headers = String (top row of CSV)
wins(3) = AIWins

FOR row = 1 TO 3
      Input to wins(row).difficulty
      Input to wins(row).totalWins

IF AIDepth = 1 THEN
      wins(0).totalWins += 1
      wins(0).difficulty = "Easy"

ELSEIF AIDepth = 3 THEN
      wins(1).totalWins += 1
      wins(1).difficulty = "Medium"

ELSE
      wins(2).totalWins += 1
      wins(2).difficulty = "Hard"

Rewrite headers to file
Rewrite wins array to file
```

Using the total number of games for each difficulty and the number of AI wins for each difficulty, the system can calculate the win rate for the AI on each difficulty.



```
Difficulty,Wins,Total Games
Easy,0,0
Medium,0,0
Hard,0,0
```

Managing All Potential Move Locations (AI Player)

I feel that one possible way of providing the AI with access to all potential move locations could be by creating another list within the program which will contain all the available moves, with all 24 moves present at the beginning of the game and as each line is placed on to the grid, the corresponding move is removed from the list. This will allow the minimax to iterate through each potential move. This will also ensure that the AI player doesn't overwrite any lines.

```
FOR each line in availableMoves
    If line start coords = newly placed line start coords AND line end
    coords = newly placed line end coords THEN
        removeIndex = index of line

availableMoves.RemoveAt removeIndex
```

Testing Strategy

Input Validation, Functions and Processes

The following validation and general function tests will allow me to identify and ensure each user input is appropriately handled, and will check each function and procedure works as it should. I will test the system using typical, acceptable data, and erroneous data that should be managed and ignored.

Test Number	Description	Data Type	Detail & Expected Result	Actual Result	Pass/Fail
1	Selecting a button on the difficulty select screen will pass in the relevant number of depth levels	Typical	Selecting Easy passes in a depth of 1, Medium a depth of 3 and Hard a depth of 5		
2	Dots should display in a clear 4x4/5x5/6x6 manner on the screen	Typical	At the beginning of the game, the dots should be drawn to the panel and their coordinates recorded		
3	The system will get the user's mouse click and display the line accordingly	Typical	User clicks a location between 2 dots, line is placed		
4	The system will ignore any inaccurate mouse clicks	Erroneous	User clicks an ambiguous location, no line is placed and turn remains the same until an appropriate line is placed		
5	The system will visually display a box to the user if a 4th line is placed on a box	Typical	User finishes a box, box is labelled P1 in the appropriate place		
6	The system will label 2 boxes simultaneously if a line is placed in the centre of two 3-line boxes	Typical	User places a line between 2 unfinished boxes, both are claimed at once		
7	Game will end once all boxes are created	Typical	If it's the user's last turn, game will end and a message box will display, showing the winner and scores		
8	System will ignore any clicks where there is already a line	Erroneous	If user tries to place a line where a line already exists, it should be ignored		

Test Number	Description	Data Type	Detail & Expected Result	Actual Result	Pass/Fail
9	System will ask if the user wants to play again at the end of a game	Typical	If user selects yes, difficulty select form is shown and game restarts. If user selects no, game closes		
10	Where a line is placed, system will record its location and name in a list of all moves	Typical	E.g. placing a line between dots 1 and 2 stores the line name as 1-2, their coordinates and the player who made the line (1/2)		
11	Where a box is made, this increases the score	Typical	Adds to player score counts depending on the number of boxes claimed by either player		
12	Where a player creates a box, they receive another turn	Typical	Placing a fourth line on a box results in this player, AI or not, placing more moves after this until a move is a normal (non-box) move		
13	For specified depth level, minimax function should only search possible turns up to this depth level	Typical	E.g. for a depth level of 3, the minimax function should only call itself and search through the layers a maximum of 3 times		
14	When minimax determines a move, this line should be placed accordingly on the grid	Typical	Move returned out of the minimax is placed on the grid in the correct position as a blue line		
15	Where move is a winning move (completes a box), AI should typically move here	Typical	A 3 lined incomplete box being present on the AI's turn should lead to the minimax placing this line here		
16	Selecting "View AI Scores" should show AI scores appropriately	Typical	When the user wants to view AI scores from the difficulty select menu, this should read from the CSV file correctly and display the total number of AI wins for easy, medium and hard difficulties out of the total number of games played for that difficulty level		
17	If AI wins, score in CSV for the selected difficulty should update on top of total number of games	Typical	AI wins: score in text file and on AIStats form increments by 1, as does total game count for that difficulty		
18	Total number of games played for selected difficulty is updated in CSV even when AI doesn't win	Typical	AI doesn't win: total games in CSV and on AIStats form increases by 1 for that difficulty, but does not increment total wins		
19	When the game is replayed, everything should be completely reset	Typical	System should empty all boxes, moves, available moves, scores and relevant variables reset to default (0)		

Testing the Minimax

To test the minimax algorithm, I will conduct some black box and white box testing, as well as some simpler tests (table above). In the black box testing I intend to run the game and explain/determine the behaviour of the AI (on a depth > 1 to showcase its recursive nature). With white box testing, I could analyse the scoring of game states, for example where there are 3 lines on a box to ensure that the minimax will recognise it should place a fourth line (scoring it highly), and situations where there are 2 lines on a box to see if the minimax knows that placing a third line on a box will result in the other player (user) being able to win a box and thus producing a worse score. I want to analyse the heuristic specifically to go into detail with how it scores moves for different game states.

Beta Testing

In order to beta test my system, I will prepare some questions for my third party, a friend who will test how well and how accurately the game functions in relation to the original game, how effective the different difficulty levels are and how challenging the AI is, and how simple it is to play the game and access all of its features. She should play through the game at least once on each difficulty level.

1. Did you find any obvious issues with the program? (Y/N)

2. If so, explain what issues you ran into or otherwise detail any bugs/annoyances

3. How clearly laid out was the interface? Was everything structured and easy to understand?

4. Did you feel anything about the program was unnecessary? Give any details

5. Did each difficulty level adequately provide a different level of difficulty?

6. How closely did the game resemble the original? Would you describe this game as being accurate enough?

7. What in particular did you like about the game?

8. What do you think could be improved to make it better?

White Box Testing

I will use white box testing to provide further evidence that the key algorithms, including the minimax/heuristic, in the game work correctly. Key algorithms I want to test will include:

- ensuring mouse click is registered and assigned the correct values on the grid, checking the mouse click coordinate values and checking these against the code

- ensuring minimax and heuristic functions appropriately decide which moves are better than others: e.g. a move completing a box has a higher priority than one that does not, boxes with 3 lines should be seen as negative for the AI player whereas standalone moves or moves with one parallel/adjacent line should be fairly neutral in scoring. Minimax on higher depths than 1 (medium/hard) should keep track of a cumulative total of scores for each possible game path to judge what the best move could be

Testing

I have produced a video that showcases the following tests, with their timestamps also given below.

Here is the link to my video: **https://youtu.be/H2Mk-ypgyHU**

Test Number	Description	Data Type	Detail & Expected Result	Actual Result	Pass/Fail	Video Timestamp
---	Program demo					**0:00**
1	Selecting a button on the difficulty select screen will pass in the relevant number of depth levels	Typical	Selecting Easy passes in a depth of 1, Medium a depth of 3 and Hard a depth of 5	Easy set AIDepth to 1, Medium set AIDepth to 3, Hard set AIDepth to 5	Pass	**1:36**
1a)	Not selecting a grid size before choosing a difficulty should be handled	Erroneous	If no grid size is selected from the drop down and a difficulty button is clicked, an error message is displayed until a grid size is selected	Error message was presented; see **screenshot 1**	Pass	**2:08**
2	Dots should display in a clear 4x4/5x5/6x6 manner on the screen	Typical	At the beginning of the game, the appropriate amount of dots should be drawn to the panel and their coordinates recorded	Selecting 4x4 presented 4x4 dots, selecting 5x5 presented 5x5 dots, selecting 6x6 presented 6x6 dots	Pass	**2:23**
3	The system will get the user's mouse click and display the line accordingly	Typical	User clicks a location between 2 dots, line is placed	Mouse was clicked between dots 8 and 9, vertical line was placed; mouse clicked between dots 20 and 26, horizontal line was placed	Pass	**2:48**
4	The system will ignore any inaccurate mouse clicks	Erroneous	User clicks an ambiguous location, no line is placed and turn remains the same until an appropriate line is placed	Mouse was clicked in white spaces not between any 2 dots; no lines were placed and player turn was not altered	Pass	**3:08**
5	The system will visually display a box to the user if a 4th line is placed on a box	Typical	User finishes a box, box is labelled P1 in the appropriate place; AI player finishes a box, box is labelled P2 in the appropriate place	When the AI player claimed a box, it was labelled with a blue "P2". When player 1 claimed a box, it was labelled with a red "P1"	Pass	**3:29**
6	The system will label 2 boxes simultaneously if a line is placed in the centre of two 3-line boxes	Typical	User/AI places a line between 2 unfinished boxes, both are claimed at once	AI claimed two boxes simultaneously and both were labelled	Pass	**4:00**
7	Game will end once all boxes are created	Typical	Game will end and a message box will display, showing the appropriate winner and scores once there are no more moves	Game ended, asked to play again on all grid sizes, when no lines were left	Pass	**4:28**
7a)	Game displays draw when scores are even	Typical	Game informs user that they drew if the sum total of boxes is the same for both players (5x5 grid)	Appropriate message was displayed: see **screenshot 2**	Pass	**5:41**

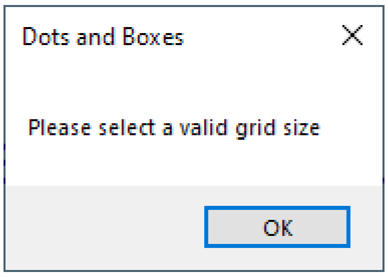
Test Number	Description	Data Type	Detail & Expected Result	Actual Result	Pass/Fail	Video Timestamp
8	System will ignore any clicks where there is already a line	Erroneous	If user tries to place a line where a line already exists, it should be ignored	Attempted to overwrite AI's moves of 3-4 and 3-7, own move of 2-6; no moves placed	Pass	**6:09**
9	System will ask if the user wants to play again at the end of a game	Typical	If user selects yes, difficulty select form is shown and game restarts. If user selects no, game closes	Yes selected; user brought back to difficulty select. No selected; program closed. Dialog box: see **screenshot 3**	Pass	**6:34**
10	Where a line is placed, system will record its location and name in a list of all moves	Typical	E.g. placing a line between dots 1 and 2 stores the line name as 1-2, their coordinates and the player who made the line (1/2)	Move 5-9 and its coordinates were added as type "Move" to the list of all moves on the grid, lines	Pass	**7:11**
11	Where a box is made, this increases the score	Typical	Adds to player score counts depending on the number of boxes claimed by either player	Boxes made by both players, scores appropriately increased	Pass	**7:58**
12	Where a player creates a box, they receive another turn	Typical	Placing a fourth line on a box results in this player, AI or not, placing more moves after this until a move is a normal (non-box) move	As a player claimed a box, they could take another turn until their move was a "normal" move (didn't claim a box)	Pass	**8:54**
13	For specified depth level, minimax function should only search possible turns up to this depth level	Typical	E.g. for a depth level of 3, the minimax function should only call itself and search through the layers a maximum of 3 times	Minimax provided with depth of 1, 3 or 5 appropriately for easy, medium or hard	Pass	**10:06**
14	When minimax determines a move, this line should be placed accordingly on the grid	Typical	Move returned out of the minimax is placed on the grid in the correct position as a blue line	Final move of 5-6 was returned out of the minimax, this move was then placed on the grid	Pass	**10:55**
15	Where there is a 2-lined box, AI should avoid placing a 3rd line as much as possible	Typical	Trying to get the minimax to place a 3rd line on a box should fail to work; it should avoid placing a 3rd line	Tried to get the AI to place a 3rd line on many 2-lined boxes: no attempts worked	Pass	**11:46**
15a)	Where move is a winning move (completes a box), AI should typically move here	Typical	A 3 lined incomplete box being present on the AI's turn should lead to the minimax placing this line here	Placed a series of 3-lined boxes, AI continued to successfully complete these boxes	Pass	**12:35**

Test Number	Description	Data Type	Detail & Expected Result	Actual Result	Pass/Fail	Video Timestamp
16	Selecting "View AI Scores" should show AI scores appropriately	Typical	When the user wants to view AI scores from the difficulty select menu, this should read from the CSV file correctly and display the total number of AI wins for easy, medium and hard difficulties out of the total number of games played for that difficulty level	Clicking View AI Scores gave a table displaying the number of AI wins out of the total games for that difficulty, as well as the win rate, that match the contents of the CSV file	Pass	**13:15**
17	If AI wins, score in CSV for the selected difficulty should update on top of total number of games	Typical	AI wins: score in text file and on AIStats form increments by 1, as does total game count for that difficulty	Wins for easy were 31/59, AI won a game on easy, this updated to 32/60	Pass	**14:01**
18	Total number of games played for selected difficulty is updated in CSV even when AI doesn't win	Typical	AI doesn't win: total games in CSV and on AIStats form increases by 1 for that difficulty, but does not increment total wins	Wins for easy were 32/60, AI lost a game on easy, this updated to 32/61	Pass	**14:59**
19	When the game is replayed, everything should be completely reset	Typical	System should empty all boxes, moves, available moves, scores and relevant variables reset to default (0)	All recorded lines, boxes and dots were cleared ready for a new game when player chose to play again	Pass	**15:53**
20	Clicking the back button on the AI Stats or main game forms should return the player back to the difficulty select form	Typical	Player is returned to difficulty select form, any moves made on the grid on the main form are reset for a new game to be played	Back buttons all directed user back to difficulty select form	Pass	**16:57**
21	Clicking the exit button on the main or difficulty select form closes the entire program	Typical	Program should completely stop running upon clicking exit on either the difficulty select or main forms	Both exit buttons entirely closed the program	Pass	**17:27**
22	**Black box testing**: gameplay on medium (depth 3)					**17:48**
23	**White box testing**: minimax on medium (depth 3)					**22:27**
24	**White box testing**: heuristic values for different game states					**35:35**
25	**White box testing**: mouse click procedure and ensuring whether clicks are/are not valid					**47:45**

Screenshots

Screenshot 1

Error message dialog box.

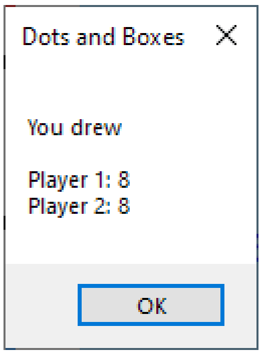


Dots and Boxes

Please select a valid grid size

Screenshot 2

This dialog box was displayed due to both player scores being equal.



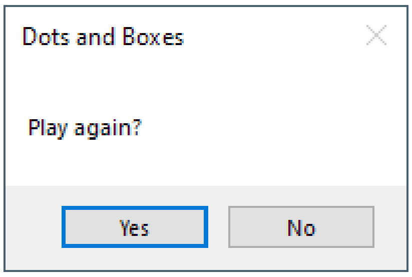
Dots and Boxes

You drew

Player 1: 8
Player 2: 8

Screenshot 3

Dialog box shown at the end of a game.

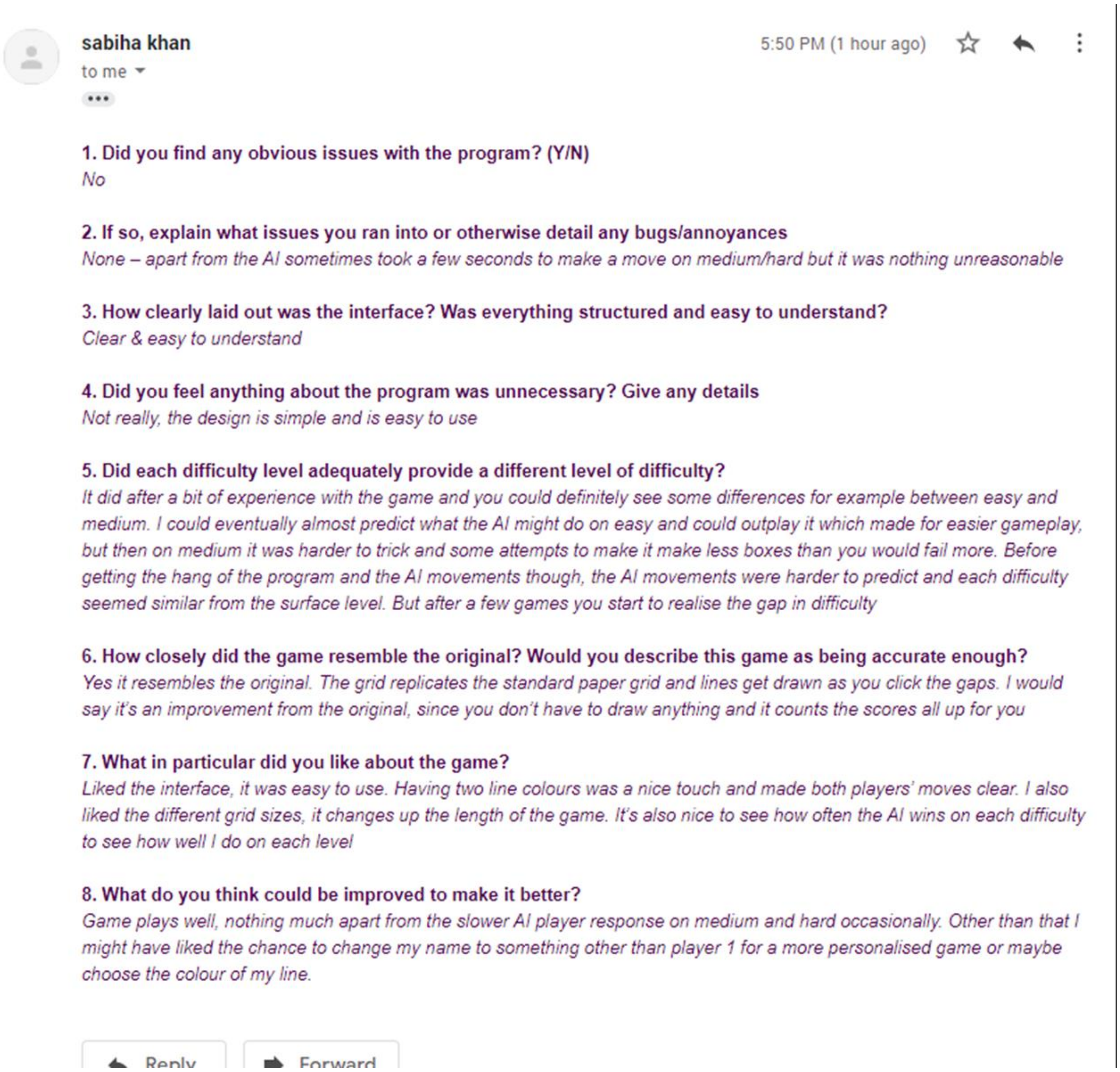


Dots and Boxes

Play again?

Third Party (Beta Test) Feedback

I asked my third party to beta test the program and play a few games on each difficulty level. I then sent her my short feedback questionnaire along with the program, and these were my responses.



sabiha khan

to me

5:50 PM (1 hour ago)

1. Did you find any obvious issues with the program? (Y/N)

No

2. If so, explain what issues you ran into or otherwise detail any bugs/annoyances

None – apart from the AI sometimes took a few seconds to make a move on medium/hard but it was nothing unreasonable

3. How clearly laid out was the interface? Was everything structured and easy to understand?

Clear & easy to understand

4. Did you feel anything about the program was unnecessary? Give any details

Not really, the design is simple and is easy to use

5. Did each difficulty level adequately provide a different level of difficulty?

It did after a bit of experience with the game and you could definitely see some differences for example between easy and medium. I could eventually almost predict what the AI might do on easy and could outplay it which made for easier gameplay, but then on medium it was harder to trick and some attempts to make it make less boxes than you would fail more. Before getting the hang of the program and the AI movements though, the AI movements were harder to predict and each difficulty seemed similar from the surface level. But after a few games you start to realise the gap in difficulty

6. How closely did the game resemble the original? Would you describe this game as being accurate enough?

Yes it resembles the original. The grid replicates the standard paper grid and lines get drawn as you click the gaps. I would say it's an improvement from the original, since you don't have to draw anything and it counts the scores all up for you

7. What in particular did you like about the game?

Liked the interface, it was easy to use. Having two line colours was a nice touch and made both players' moves clear. I also liked the different grid sizes, it changes up the length of the game. It's also nice to see how often the AI wins on each difficulty to see how well I do on each level

8. What do you think could be improved to make it better?

Game plays well, nothing much apart from the slower AI player response on medium and hard occasionally. Other than that I might have liked the chance to change my name to something other than player 1 for a more personalised game or maybe choose the colour of my line.

Evaluation

Checking Against Requirements

Requirement Number	Detail	Is Requirement Met?	Testing Cross-Reference	Summary
1	Create a game board of dots, e.g. 4x4, where the user is allowed to interact with it	Yes	Test 2	Depending on whether a 4x4, 5x5 or 6x6 grid is selected, a grid of this size of dots is created
1.1	Dots an even space apart	Yes	Test 2	Gaps between dots are calculated with attribute gapSize
1.2	Store each dot as a coordinate on the grid	Yes	Test 2	Stored within object Dot for all dots on the grid as X and Y coordinates
2, 2.1	User selects difficulty: selection between easy, medium and hard	Yes	Test 1	Difficulty select form allows user to choose a difficulty at the beginning of every game
2.2	Adjust number of scenarios minimax algorithm will iterate through based on difficulty selected	Yes	Test 1	Maximum search depth is set accordingly for each difficulty: depth 1 for easy, depth 3 for medium, depth 5 for hard
3, 3.2	Have a systematic turn-based system that lets both players have a turn one after the other; on a normal move, switch turns after each player moves	Yes	Program demo	Player turns are alternated each normal turn. If player 1 moves, player 2 then moves afterwards. If it's the AI player's go on the first move of the game, a random move is generated at the beginning
3.1	Check if last move completed a box. If it did, player receives another turn until a normal move is made	Yes	Test 12	Completing a box gives the player another turn until a move no longer completes a box, in which the turns are alternated as normal. Similarly if the AI player completes a box they can do the same
4, 4.3	Enable the user to place lines on the game board and display these lines visually on the board; display line on grid in correct colour and	Yes	Test 3	Lines are placed based on user's click input in red, and placed based on the Move object returned from the minimax for the AI in blue in the correct locations

	relative to where the user clicked			
4.1	Accept user's click location as location to place line	Yes	Test 3, white box testing of click sub	User click coordinates are translated into the game space, calculating whether the line should be horizontal, vertical or invalid and displaying/not displaying it appropriately
4.2	Ensure click is ignored if it does not correlate accurately enough to a space on the grid/overwrites another line	Yes	Test 4, test 8	No move is made and no further turns are made if the player attempts to overwrite an existing move or click an empty space
4.4	Record all line positions on grid	Yes	Test 10	All line coordinates are stored within their object type, Move, and stored in a list of all moves
5	Have the scores for both players displayed to the side for the user to see at all times	Yes	Program demo	Player 1's score is displayed in red to the left, player 2's score is displayed in blue to the right
6, 6.1	Increment the score if a box is made; display updated score each turn	Yes	Test 11	As player 1 or 2 claims a box, this score is incremented appropriately with the number of boxes claimed being added to the current score. This is displayed within the labels on the left and right
7	Display the claimed boxes visually	Yes	Test 5, test 6	When player 1 claims a box, this box is labelled as a red P1. When player 2 claims a box, this is labelled as a blue P2

8	Have a functioning AI player that can effectively take into account all the moves both it and the user can make with each turn, making a decision via an implemented minimax algorithm	Yes	Test 14, test 15, test 15a, white box testing of minimax/heuristic	Current game state, available moves as well as which player, the maximum depth and alpha/beta are passed into the minimax on its initial call. The minimax is recursively called, decreasing the depth with each call and alternating between player 1 (minimising) and player 2 (maximising), first acquiring leaf nodes and appropriately getting the greatest/smallest heuristic score on each level depending on whether it is minimising or maximising and accumulating a total score for each move path up to the top layer to decide on the best move to make
8.1	AI iterates through fewer/more depth levels of the minimax depending on the difficulty, to alter how far it can think ahead and thus how "intelligent" it is	Yes	Test 13	With the passed in depth level from the main game, AIDepth, which is set to 1, 3 or 5 depending on the difficulty, the minimax will decrease from this value until it reaches 0 and therefore hits the leaf nodes. The minimax does not search beyond the number of layers it is passed in
8.2	Record and update all moves available to AI player each turn	Yes	White box testing of minimax	The availableMoves list in the main game is removed from as either player makes moves to ensure the AI cannot attempt to use these moves or take them into consideration. tempAvailableMoves also exists in the minimax to serve as a temporary version of this list when it iterates through theoretical game states

9	End the game when the board has been filled	Yes	Test 7	When there are no moves left, the game displays a message box showing the winner and scores
9.1	Ask if the user wants to replay	Yes	Test 9, test 19	User is given the option to replay, which returns them to the difficulty select screen for another game, or to quit, which quits the program
10	Calculate and display the overall winner of the game	Yes	Test 7, test 7a	At the end of the game, the winner is determined by judging which player's score was larger. If they were equal a message indicating a draw is displayed instead. Scores are displayed underneath
10.1	If the AI player won, add this to total games won in another file for corresponding difficulty, having this data accessible within the system. If the AI player lost, only update total games count	Yes	Test 16, test 17, test 18	The CSV file is written to at the end of every game. Regardless of which player won, the Total Games column is updated on the associated difficulty row by incrementing the amount by 1 in the file. The Wins column is incremented by 1 on the associated difficulty row only when player 2 was the winner and there was no draw. The win rate is added as an additional column on the AI stats form, calculated within the program, using the total games and won games

Personal Evaluation

Overall, I think my system successfully meets my list of requirements. As for my investigation, I managed to find out that an AI player can play a turn-based game like dots and boxes, using a minimax algorithm to pose as an appropriate challenge to the player. The UI is clear and the program overall is simple to use.

Using the feedback from my beta testing (see testing) and through my own testing, I do think my system emulates the traditional gameplay of dots and boxes well, additionally introducing newer features such as extra grid sizes and having a clear user interface. The AI player successfully provides different levels of challenge for the user, though perhaps the gap in difficulty is less obvious to a less experienced player who may not have developed much of a strategy – however I do agree that sometimes the AI can be a bit slower to respond compared to easy difficulty, especially when there are several potential successive boxes to claim. This is more of a technical issue with the recursive nature of the minimax. With other games this is avoidable by stopping the search when the game is won, but the multiple turns rule in dots and boxes makes this much more complicated. Personalisable names/colours could have been interesting from a player's perspective, even though it wouldn't affect the core gameplay.

In order to reduce the complexity and therefore possibly help with the program's performance, maybe I could have used a different variant of the minimax, negamax, where instead of alternating between min and max it combines the two into one function, calling negamax and -negamax recursively depending on whether it is player 1/2. It's based on the principle that the opponent is also playing perfectly, and will therefore play the inverse of the maximum move (hence -negamax). It's a more refined version of the minimax and may have helped with the speed of my program.

On top of this, maybe I could have added a feature like saving the game. It might be a bit excessive for a game of this nature however I think it could be an interesting feature to be able to save your progress and come back to the game later. It could have added a layer of complexity to the project.

Technical Solution

In my technical solution, I finalised my decision to include other board sizes. This means that on top of a 4x4 grid, there is also the option of a 5x5 and 6x6 grid in order to add some variety and give the user some choice as to how long they want the game to be. Additionally, in a 5x5 grid, there is the possibility of a draw due to the number of total dots being odd, providing another end game scenario.

Main Form (Dots and Boxes)

The main game form where most of the processing takes place. Dots, lines, and boxes are stored and managed, scores are kept, objects are displayed, and the gameplay including the minimax for the AI is found here. It additionally updates the AI scores in the file for the AI Stats form to use.

```
Imports System.IO

Public Class Game
    Private Const windowHeight = 600
    Private Const windowWidth = 1000
    Private gridSize As Integer
    Private Const panelHeight = 400
    Private Const panelWidth = 400
    Private gapSize As Decimal
    Private dots As New List(Of Dot)
    Private lines As New List(Of Move) 'all moves made (game state)
    Private availableMoves As New List(Of Move) 'all moves available
    Private boxes As New List(Of Box)
    Private drawFormat As StringFormat
    Private Const dotSize As Integer = 10
    Private P1Score As Integer
    Private P2Score As Integer
    Private playerTurn As Integer
    Private boxCount As Integer
    Private AIdepth As Integer
    Public AIWins(2) As AIWin
    Public fileHeaders As String
    Private fileName As String = "aiscores.csv"

    Private Sub CreateGameGrid()
        Dim xcoord As Integer
        Dim ycoord As Integer
        gapsize = Math.Floor(((1 / gridSize) * panelHeight))
        For x = 0 To gridSize - 1 'Embedded loop that sets coordinates of all dots and adds them to a list of point values
        For y = 0 To gridSize - 1
            xcoord = CInt((panelWidth / dotSize) + (gapSize * x))
            ycoord = CInt((panelHeight / dotSize) + (gapSize * y))

            dots.Add(New Dot(New Point(xcoord, ycoord), (dots.Count + 1).ToString))
        Next
    Next

End Sub

Private Sub SetGridSize()
    gridSize = DifficultySelect.gridDimension 'Sets the grid size to 4x4/5x5/6x6 as chosen by the user
End Sub


Private Sub DrawBoard(sender As Object, e As PaintEventArgs) Handles GameBoard.Paint
    Dim coordinates As New List(Of Point)
    Dim gameSpace As Panel = DirectCast(sender, Panel) 'Sets the panel to the variable gameSpace
    Dim gridGraphics As Graphics = gameSpace.CreateGraphics
    Dim StartDrawPos As Point
    Dim EndDrawPos As Point
    Dim lineDraw As Pen
    Dim drawBrush As SolidBrush
    Dim gameEndString As String = vbCrLf & vbCrLf & "Player 1: " & P1Score & vbCrLf & "Player 2: " & P2Score

    Me.Height = windowHeight 'Setting everything visual up
    Me.Width = windowWidth
    gameSpace.BackColor = Color.White
    gameSpace.Height = panelHeight
    gameSpace.Width = panelWidth
    gameSpace.Location = New Point((windowWidth - panelWidth) / 2, (windowHeight - panelHeight) / 2) 'Centres the panel on to the window
    lbl_P1Score.Font = New Drawing.Font("Arial", 20)
    lbl_P2Score.Font = New Drawing.Font("Arial", 20)
    lbl_WhoseTurn.Font = New Drawing.Font("Arial", 24)
    lbl_P1Score.ForeColor = Color.Red
    lbl_P2Score.ForeColor = Color.Blue
    lbl_WhoseTurn.ForeColor = Color.Black
    lbl_P1Score.Location = New Point(gameSpace.Left - lbl_P1Score.Width - 50, windowHeight / 2)
    lbl_P2Score.Location = New Point(gameSpace.Right + 50, windowHeight / 2)
    lbl_WhoseTurn.Location = New Point(gameSpace.Location.X * 1.3, gameSpace.Top - 50)
    btn_Back.Location = New Point(gameSpace.Location.X, gameSpace.Location.Y + panelHeight + 20)
    btn_Exit.Location = New Point(btn_Back.Location.X + 100, btn_Back.Location.Y)
```

```
drawBrush = New SolidBrush(Color.Black)
lineDraw = New Pen(Color.Black, 5)
drawFormat = New StringFormat()
drawFormat.Alignment = StringAlignment.Center
drawFormat.LineAlignment = StringAlignment.Center

If lines.Count <= 1 Then 'Gets AI depth as selected by user at beginning of the game
    SetAIDepth()
End If

For Each dot In dots 'Draws all of the dots
    gridGraphics.FillEllipse(drawBrush, dot.GetX, dot.GetY, dotSize, dotSize)
Next

For Each line In lines 'Draws all of the lines
    StartDrawPos = line.GetStartPos
    EndDrawPos = line.GetEndPos

    lineDraw.Color = line.GetLineColour

    If StartDrawPos.Y = EndDrawPos.Y Then
        gridGraphics.DrawLine(lineDraw, StartDrawPos.X + 10, StartDrawPos.Y + 5, EndDrawPos.X + 1, EndDrawPos.Y + 5)

    ElseIf StartDrawPos.X = EndDrawPos.X Then
        gridGraphics.DrawLine(lineDraw, StartDrawPos.X + 5, StartDrawPos.Y + 10, EndDrawPos.X + 5, EndDrawPos.Y + 1)
    End If

Next

For Each box In boxes 'Draws all claimed boxes
    If boxes.Count <> 0 Then
        drawBrush.Color = box.GetLabelTextColour
        gridGraphics.DrawString(box.GetLabelText, New Drawing.Font("Arial", gapsize / 3), drawBrush, box.SetLabelPosition, drawFormat)
    End If

Next

If boxCount = (gridSize - 1) ^ 2 Then

    If GetWinner() = 1 Or GetWinner() = 2 Then
        MsgBox("Player " & GetWinner() & " wins" & gameEndString)

    Else MsgBox("You drew" & gameEndString)
```

```
        End If

        GetAIData()
        UpdateAIData()
        AskToEndOrReplayGame()

    End If
End Sub

Private Function GetWinner()
    Dim winner As Integer
    If P1Score > P2Score Then
        winner = 1
    ElseIf P2Score > P1Score Then
        winner = 2
    Else
        winner = 0
    End If

    Return winner

End Function

Public Structure AIWin
    Dim difficulty As String
    Dim totalWins As Integer
    Dim totalGames As Integer
End Structure

Private Sub GetAIData()
    '*** Read data from CSV file and update within program ***
    Dim fileLines As New List(Of String)
    Dim reader As New StreamReader(fileName)
    Dim currentLine As String

    For row = 0 To 3
        currentLine = reader.ReadLine()
        If row = 0 Then
            fileHeaders = currentLine
        Else
            fileLines.Add(currentLine)
            Dim data As String() = fileLines(row - 1).Split(",")
```

```
                AIWins(row - 1).difficulty = data(0)
                AIWins(row - 1).totalWins = CInt(data(1))
                AIWins(row - 1).totalGames = CInt(data(2))
            End If
        Next

        If AIdepth = 1 Then
            If GetWinner() = 2 Then
                AIWins(0).totalWins += 1
            End If
            AIWins(0).totalGames += 1
        ElseIf AIdepth = 3 Then
            If GetWinner() = 2 Then
                AIWins(1).totalWins += 1
            End If
            AIWins(1).totalGames += 1
        ElseIf AIdepth = 5 Then
            If GetWinner() = 2 Then
                AIWins(2).totalWins += 1
            End If
            AIWins(2).totalGames += 1
        End If

        reader.Close()

    End Sub

    Private Sub UpdateAIData()
        '*** Write updated data to CSV ***
        Dim linesToWrite As New List(Of String)
        Dim writer As New StreamWriter(fileName)

        For i = 0 To 2
            linesToWrite.Add(AIWins(i).difficulty & "," & AIWins(i).totalWins & "," & AIWins(i).totalGames)
        Next

        For row = 0 To 3
            If row = 0 Then
                writer.WriteLine(fileHeaders)
            Else
                writer.WriteLine(linesToWrite(row - 1))
            End If
        Next
        writer.Close()

    End Sub

    Private Sub AskToEndOrReplayGame()
        Dim ExitGame As DialogResult

        ExitGame = MsgBox("Play again?", MsgBoxStyle.YesNo)
        If ExitGame = MsgBoxResult.Yes Then
            Hide()
            SetNewGame()
            DifficultySelect.Show()
        ElseIf ExitGame = MsgBoxResult.No Then
            Application.Exit()
        End If
    End Sub

    Private Function DoesNotOverwriteMove(ByVal verticalLine As Boolean, ByVal horizontalLine As Boolean, ByVal firstDotIndex As Integer)
        Dim falseFlag As Boolean = False

        If verticalLine Then
            For line = 0 To lines.Count - 1
                If lines(line).GetStartPos = dots(firstDotIndex).GetXY And lines(line).GetEndPos = dots(firstDotIndex + 1).GetXY Then
                    falseFlag = True
                End If
            Next

        ElseIf horizontalLine Then
            For line = 0 To lines.Count - 1
                If lines(line).GetStartPos = dots(firstDotIndex).GetXY And lines(line).GetEndPos = dots(firstDotIndex + gridSize).GetXY Then
                    falseFlag = True
                End If
            Next

        End If

        If falseFlag Then
            Return False

        Else Return True

        End If
    End Function

    Private Sub CheckLines(ByVal sender As Object, ByVal e As System.Windows.Forms.MouseEventArgs) Handles GameBoard.MouseClick
        Dim mouseCoords As New Point(e.X, e.Y)
        Dim errorRange As Integer = gapsize / 5 'Margin of error allowed for the user's mouse click
        Dim vertical As Boolean
        Dim horizontal As Boolean
        Dim validMove As Boolean
        Dim nextDotIndex As Integer
        Dim numOfBoxesToMake As Integer
        Dim TBox, BBox, LBox, RBox As Boolean
        Dim currentIndex As Integer
        Dim lineFound As Boolean
        Dim AIMove As Move

        For dotVal = 0 To (gridSize ^ 2) - 1
            vertical = False
            horizontal = False
            validMove = True
            numOfBoxesToMake = 0
            lineFound = False
            TBox = False
            BBox = False
            LBox = False
            RBox = False

            If playerTurn = 1 Then
                Try 'Check click correlates to a position
                    If (mouseCoords.Y > dots(dotVal).GetY + errorRange And mouseCoords.Y < dots(dotVal + 1).GetY - errorRange) And ((mouseCoords.X
<= dots(dotVal).GetX And mouseCoords.X > dots(dotVal).GetX - errorRange) Or (mouseCoords.X >= dots(dotVal).GetX And mouseCoords.X <
dots(dotVal).GetX + errorRange)) Then
                        vertical = True
                        nextDotIndex = 1

                    ElseIf (mouseCoords.X > dots(dotVal).GetX + errorRange And mouseCoords.X < dots(dotVal + gridSize).GetX - errorRange) And
((mouseCoords.Y <= dots(dotVal).GetY And mouseCoords.Y > dots(dotVal).GetY - errorRange) Or (mouseCoords.Y >= dots(dotVal).GetY And mouseCoords.Y <
dots(dotVal).GetY + errorRange)) Then
                        horizontal = True
                        nextDotIndex = gridSize

                    End If
                Catch ex As Exception

                End Try

            ElseIf playerTurn = 2 Then

                If availableMoves.Count = 1 Then
                    AIMove = availableMoves(0)
                    If AIMove.GetOrientation = "H" Then
                        horizontal = True
                    Else vertical = True
                    End If

                    dotVal = CInt(AIMove.GetLeftName) - 1

                ElseIf availableMoves.Count > 1 Then
                    ComputerMove(AIMove)
                    If AIMove.GetOrientation = "H" Then
                        horizontal = True
                    Else vertical = True
                    End If

                    dotVal = CInt(AIMove.GetLeftName) - 1

                End If

            End If

            If DoesNotOverwriteMove(vertical, horizontal, dotVal) = False Then
                validMove = False

            End If

            If validMove And (horizontal Or vertical) Then

                If playerTurn = 1 Then
                    lines.Add(New Move(dots(dotVal), dots(dotVal + nextDotIndex), 1))
                Else
                    lines.Add(New Move(New Dot(AIMove.GetStartPos, AIMove.GetLeftName), New Dot(AIMove.GetEndPos, AIMove.GetRightName), 2))
                End If

                currentIndex = lines.Count - 1
```

```
            RemoveMove(availableMoves, lines, currentIndex)

            numOfBoxesToMake = CheckForBox(lines(currentIndex), lines, TBox, BBox, LBox, RBox)

            Select Case numOfBoxesToMake
                Case 1
                    boxCount += 1
                Case 2
                    boxCount += 2
            End Select

            If numOfBoxesToMake > 0 Then
                If horizontal Then

                    CreateBoxH(lines(currentIndex).GetLeftName, lines(currentIndex).GetRightName, numOfBoxesToMake, TBox, BBox)

                ElseIf vertical Then

                    CreateBoxV(lines(currentIndex).GetLeftName, lines(currentIndex).GetRightName, numOfBoxesToMake, LBox, RBox)

                End If
            End If
            SetPlayerTurn(numOfBoxesToMake)
            UpdateLabels()
            GameBoard.Refresh() 'Refreshes board to add newly drawn objects (lines, boxes)

        End If

    Next

End Sub



Private Sub SetPlayerTurn(ByVal numBoxes As Integer)
    If numBoxes = 0 Then
        If playerTurn = 1 Then
            playerTurn = 2
        Else playerTurn = 1

        End If
    Else
        If playerTurn = 1 Then
```

```
                P1Score = P1Score + numBoxes
            Else P2Score = P2Score + numBoxes

            End If

        End If
    End Sub

    Private Sub UpdateLabels()
        lbl_P1Score.Text = "Player 1" & vbCrLf & P1Score
        lbl_P2Score.Text = "Player 2" & vbCrLf & P2Score
        lbl_WhoseTurn.Text = "Player " & playerTurn & "'s turn"
    End Sub

    Private Sub RemoveMove(ByRef movesAvailable As List(Of Move), ByVal movesMade As List(Of Move), ByVal moveIndex As Integer)
        Dim lineFound As Boolean = False
        Dim removeIndex As Integer

        For Each line In movesAvailable
            If line.GetStartPos = movesMade(moveIndex).GetStartPos And line.GetEndPos = movesMade(moveIndex).GetEndPos Then
                removeIndex = movesAvailable.IndexOf(line)
                lineFound = True

            End If
        Next

        If lineFound Then
            movesAvailable.RemoveAt(removeIndex)
        End If
    End Sub

    Private Function BoxesMade(ByVal potentialMove As Move, ByVal movesList As List(Of Move))
        Return CheckForBox(potentialMove, movesList, Nothing, Nothing, Nothing, Nothing)

    End Function

    'direction 1 = up or left, direction 2 = down or right
    Private Function checkNumLines(ByVal potentialMove As Move, ByVal movesList As List(Of Move), ByVal direction As Integer)
        Dim lineL, lineR, lineB, lineA As Boolean
        Dim currentLineName As String
        Dim currentLineLeftPart, currentLineRightPart As Integer

        currentLineName = potentialMove.GetLineName
```

```
currentLineLeftPart = potentialMove.GetLeftName
currentLineRightPart = potentialMove.GetRightName

If potentialMove.GetOrientation = "H" And direction = 2 Then
    If (currentLineLeftPart Mod gridSize) <> 0 Then
        For Each line In movesList
            If currentLineLeftPart = line.GetLeftName And currentLineRightPart = line.GetRightName + (gridSize - 1) Then
                lineL = True
            ElseIf currentLineLeftPart = line.GetLeftName - gridSize And currentLineRightPart = line.GetRightName - 1 Then
                lineR = True
            ElseIf currentLineLeftPart = line.GetLeftName - 1 And currentLineRightPart = line.GetRightName - 1 Then
                lineB = True
            End If
        Next
    End If


    If lineL And lineR And lineB Then
        Return 0
    ElseIf (lineL And lineR And Not lineB) Or (lineL And lineB And Not lineR) Or (lineR And lineB And Not lineL) Then
        Return 3
    ElseIf (lineL And Not (lineR Or lineB)) Or (lineR And Not (lineL Or lineB)) Or (lineB And Not (lineL Or lineR)) Then
        Return 2
    Else Return 1
    End If

    lineL = False
    lineR = False

ElseIf potentialMove.GetOrientation = "H" And direction = 1 Then
    If currentLineLeftPart Mod gridSize <> 1 Then
        For Each line In movesList
            If currentLineLeftPart = line.GetLeftName + 1 And currentLineRightPart = line.GetRightName + gridSize Then
                lineL = True
            ElseIf currentLineLeftPart = line.GetLeftName - (gridSize - 1) And currentLineRightPart = line.GetRightName Then
                lineR = True
            ElseIf currentLineLeftPart = line.GetLeftName + 1 And currentLineRightPart = line.GetRightName + 1 Then
                lineA = True
            End If
        Next
    End If

    If lineL And lineR And lineA Then
```

```
                Return 0
            ElseIf (lineL And lineR And Not lineA) Or (lineL And lineA And Not lineR) Or (lineR And lineA And Not lineL) Then
                Return 3
            ElseIf (lineL And Not (lineR Or lineA)) Or (lineR And Not (lineL Or lineA)) Or (lineA And Not (lineL Or lineR)) Then
                Return 2
            Else Return 1
            End If

        ElseIf potentialMove.GetOrientation = "V" And direction = 2 Then
            If currentLineLeftPart < (gridSize ^ 2) - gridSize Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName - 1 And currentLineRightPart = line.GetRightName - gridSize Then
                        lineB = True
                    ElseIf currentLineLeftPart = line.GetLeftName And currentLineRightPart = line.GetRightName - (gridSize - 1) Then
                        lineA = True
                    ElseIf currentLineLeftPart = line.GetLeftName - gridSize And currentLineRightPart = line.GetRightName - gridSize Then
                        lineR = True
                    End If
                Next
            End If

            If lineB And lineA And lineR Then
                Return 0
            ElseIf (lineB And lineA And Not lineR) Or (lineB And lineR And Not lineA) Or (lineR And lineA And Not lineB) Then
                Return 3
            ElseIf (lineB And Not (lineA Or lineR)) Or (lineR And Not (lineA Or lineB)) Or (lineA And Not (lineL Or lineB)) Then
                Return 2
            Else Return 1
            End If

            lineA = False
            lineB = False

        ElseIf potentialMove.GetOrientation = "V" And direction = 1 Then
            If currentLineLeftPart > gridSize Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName + (gridSize - 1) And currentLineRightPart = line.GetRightName Then
                        lineB = True
                    ElseIf currentLineLeftPart = line.GetLeftName + gridSize And currentLineRightPart = line.GetRightName + 1 Then
                        lineA = True
                    ElseIf currentLineLeftPart = line.GetLeftName + gridSize And currentLineRightPart = line.GetRightName + gridSize Then
                        lineL = True
                    End If
                Next
            End If

            If lineA And lineB And lineL Then
                Return 0
            ElseIf (lineA And lineB And Not lineL) Or (lineB And lineL And Not lineA) Or (lineL And lineA And Not lineB) Then
                Return 3
            ElseIf (lineB And Not (lineA Or lineL)) Or (lineL And Not (lineA Or lineB)) Or (lineA And Not (lineL Or lineB)) Then
                Return 2
            Else Return 1
            End If

        End If

    End Function

    Private Function CheckForBox(ByVal potentialMove As Move, ByVal movesList As List(Of Move), ByRef TBox As Boolean, ByRef BBox As Boolean, ByRef
LBox As Boolean, ByRef RBox As Boolean)
        Dim currentLineName As String
        Dim currentLineLeftPart, currentLineRightPart As Integer
        Dim lineL, lineR, lineB, lineA As Boolean
        Dim numOfBoxesToBeMadeH, numOfBoxesToBeMadeV As Integer

        currentLineName = potentialMove.GetLineName
        currentLineLeftPart = potentialMove.GetLeftName
        currentLineRightPart = potentialMove.GetRightName

        If potentialMove.GetOrientation = "H" Then
            If (currentLineLeftPart Mod gridSize) <> 0 Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName And currentLineRightPart = line.GetRightName + (gridSize - 1) Then
                        lineL = True
                    ElseIf currentLineLeftPart = line.GetLeftName - gridSize And currentLineRightPart = line.GetRightName - 1 Then
                        lineR = True
                    ElseIf currentLineLeftPart = line.GetLeftName - 1 And currentLineRightPart = line.GetRightName - 1 Then
                        lineB = True
                    End If
                Next
            End If

            If lineL And lineR And lineB Then
                numOfBoxesToBeMadeH += 1
                BBox = True
            End If

            lineL = False
            lineR = False

            If currentLineLeftPart Mod gridSize <> 1 Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName + 1 And currentLineRightPart = line.GetRightName + gridSize Then
                        lineL = True
                    ElseIf currentLineLeftPart = line.GetLeftName - (gridSize - 1) And currentLineRightPart = line.GetRightName Then
                        lineR = True
                    ElseIf currentLineLeftPart = line.GetLeftName + 1 And currentLineRightPart = line.GetRightName + 1 Then
                        lineA = True
                    End If
                Next
            End If

            If lineL And lineR And lineA Then
                numOfBoxesToBeMadeH += 1
                TBox = True
            End If

            If numOfBoxesToBeMadeH > 0 Then
                Return numOfBoxesToBeMadeH

            End If

        ElseIf potentialMove.GetOrientation = "V" Then

            If currentLineLeftPart < (gridSize ^ 2) - gridSize Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName - 1 And currentLineRightPart = line.GetRightName - gridSize Then
                        lineB = True
                    ElseIf currentLineLeftPart = line.GetLeftName And currentLineRightPart = line.GetRightName - (gridSize - 1) Then
                        lineA = True
                    ElseIf currentLineLeftPart = line.GetLeftName - gridSize And currentLineRightPart = line.GetRightName - gridSize Then
                        lineR = True
                    End If
                Next
            End If

            If lineA And lineB And lineR Then
                numOfBoxesToBeMadeV += 1
                RBox = True
            End If

            lineA = False
            lineB = False

            If currentLineLeftPart > gridSize Then
                For Each line In movesList
                    If currentLineLeftPart = line.GetLeftName + (gridSize - 1) And currentLineRightPart = line.GetRightName Then
                        lineB = True
                    ElseIf currentLineLeftPart = line.GetLeftName + gridSize And currentLineRightPart = line.GetRightName + 1 Then
                        lineA = True
                    ElseIf currentLineLeftPart = line.GetLeftName + gridSize And currentLineRightPart = line.GetRightName + gridSize Then
                        lineL = True
                    End If
                Next
            End If

            If lineA And lineB And lineL Then
                numOfBoxesToBeMadeV += 1
                LBox = True

            End If
            If numOfBoxesToBeMadeV > 0 Then
                Return numOfBoxesToBeMadeV

            End If

            If numOfBoxesToBeMadeH = 0 And numOfBoxesToBeMadeV = 0 Then
                Return 0
            End If

            TBox = False
            BBox = False
            RBox = False
            LBox = False


        End If
    End Function

    Public Sub SetNewGame()
        Dim AIFirstMove As Move
```

```
        Dim listIndex As Integer

        P1Score = 0
        P2Score = 0
        lines.Clear()
        boxes.Clear()
        dots.Clear()
        availableMoves.Clear()
        boxCount = 0
        playerTurn = Int((2 * Rnd()) + 1) 'Randomise which player starts by generating a number between 1 and 2
        UpdateLabels()

        SetGridSize()
        CreateGameGrid()

        For dot = 0 To dots.Count - 1
            If dots(dot).GetID Mod gridSize <> 0 Then
                availableMoves.Add(New Move(dots(dot), dots(dot + 1), 0))
            End If

            If dots(dot).GetID <= ((gridSize ^ 2) - gridSize) Then
                availableMoves.Add(New Move(dots(dot), dots(dot + gridSize), 0))
            End If

        Next

        If playerTurn = 2 Then
            listIndex = Int((availableMoves.Count - 1) * Rnd()) 'Gets a random first move for the AI player if the AI player is selected to go
first
            AIFirstMove = availableMoves(listIndex)
            lines.Add(New Move(New Dot(AIFirstMove.GetStartPos, AIFirstMove.GetLeftName), New Dot(AIFirstMove.GetEndPos, AIFirstMove.GetRightName),
2))
            RemoveMove(availableMoves, lines, lines.Count - 1)
            SetPlayerTurn(0)
            UpdateLabels()
        End If

        GameBoard.Refresh()
    End Sub

    Private Sub SetAIDepth()
        AIdepth = DifficultySelect.AILevels
    End Sub
    Private Sub CreateBoxH(ByVal currentLineLeftPart As Integer, ByVal currentLineRightPart As Integer, ByVal numOfBoxesToBeMade As Integer, ByVal
TBox As Boolean, ByVal BBox As Boolean)
        Dim upperLineName As String
        Dim upperLineStart As Point
        Dim upperLineEnd As Point

        For n = 1 To numOfBoxesToBeMade
            If TBox And BBox And numOfBoxesToBeMade > 1 Then
                BBox = False
            End If

            If TBox Then
                upperLineName = (currentLineLeftPart - 1).ToString & "-" & (currentLineRightPart - 1).ToString

            ElseIf BBox Then
                upperLineName = currentLineLeftPart.ToString & "-" & currentLineRightPart.ToString

            End If


            For Each line In lines
                If line.GetLineName = upperLineName Then
                    upperLineStart = line.GetStartPos
                    upperLineEnd = line.GetEndPos
                End If
            Next

            boxes.Add(New Box(upperLineStart, upperLineEnd, playerTurn, numOfBoxesToBeMade))
            TBox = False
            BBox = True
        Next

    End Sub

    Private Sub CreateBoxV(ByVal currentLineLeftPart As Integer, ByVal currentLineRightPart As Integer, ByVal numOfBoxesToBeMade As Integer, ByVal
LBox As Boolean, ByVal RBox As Boolean)
        Dim upperLineName As String
        Dim upperLineStart As Point
        Dim upperLineEnd As Point

        For n = 1 To numOfBoxesToBeMade
            If LBox And RBox And numOfBoxesToBeMade > 1 Then
```

```
                RBox = False
            End If

            If RBox Then
                upperLineName = currentLineLeftPart.ToString & "-" & (currentLineRightPart + (gridSize - 1)).ToString

            ElseIf LBox Then
                upperLineName = (currentLineLeftPart - gridSize).ToString & "-" & (currentLineRightPart - 1).ToString

            End If

            For Each line In lines
                If line.GetLineName = upperLineName Then
                    upperLineStart = line.GetStartPos
                    upperLineEnd = line.GetEndPos
                End If
            Next

            boxes.Add(New Box(upperLineStart, upperLineEnd, playerTurn, numOfBoxesToBeMade))
            LBox = False
            RBox = True
        Next

    End Sub

    Private Sub Game_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Randomize()
        Hide()
        DifficultySelect.Show()
        GetAIData()
    End Sub

    Private Sub HideOnStartup(sender As Object, e As EventArgs) Handles Me.Shown
        Me.Visible = False
    End Sub

    Private Function Heuristic(ByVal potentialMove As Move, ByVal movesMade As List(Of Move)) 'Used within minimax to determine the score of any
given move
        Dim score As Integer

        If BoxesMade(potentialMove, movesMade) > 0 Then
            If BoxesMade(potentialMove, movesMade) = 2 Then
                score += 500
            ElseIf BoxesMade(potentialMove, movesMade) = 1 Then
                score += 250
            End If
        End If

        'Checks for the number of lines around proposed move
        For direction = 1 To 2 'Checks above and below/left and right a proposed move
            If checkNumLines(potentialMove, movesMade, direction) = 2 Then
                score += 100
            ElseIf checkNumLines(potentialMove, movesMade, direction) = 3 And BoxesMade(potentialMove, movesMade) = 0 Then
                score -= 500
            ElseIf checkNumLines(potentialMove, movesMade, direction) = 3 And BoxesMade(potentialMove, movesMade) = 1 Then 'Checks boxes can be
made consecutively, giving a score equivalent to acquiring 2 boxes simultaneously
                score += 250
            End If
        Next

        Return score

    End Function

    Private Function Minimax(ByVal player As Integer, ByVal depth As Integer, ByVal movesMade As List(Of Move), ByVal movesAvailable As List(Of
Move), ByVal currentMove As Move, ByRef bestMove As Move, ByVal alpha As Integer, ByVal beta As Integer)
        Dim tempGameState As List(Of Move)
        Dim tempAvailableMoves As List(Of Move)
        Dim score As Integer
        Dim greatest As Integer
        Dim tempPresentMove As Move
        Dim moveIndex As Integer
        Dim moveToReturn As Move

        'Creates temporary board states for searching a particular game route, which are passed in with each recursive call
        tempGameState = movesMade
        tempAvailableMoves = movesAvailable

        If depth = 0 Or tempAvailableMoves.Count = 1 Then 'If the depth reaches the given max depth or there is only one move left on the board
(end game state), it will return this move score back up the minimax
            If tempAvailableMoves.Count = 1 Then 'returns whatever move is left if this is the last available move
                Return Heuristic(tempAvailableMoves(0), tempGameState)

            Else 'returns score of leaf node passed in if depth reaches limit
                Return Heuristic(currentMove, tempGameState)
            End If
        End If

        If player = 2 Then 'Maximising
            greatest = -999999

            For i = 0 To tempAvailableMoves.Count - 1
                score = 0
                tempPresentMove = tempAvailableMoves(i)
                tempGameState.Add(New Move(New Dot(tempPresentMove.GetStartPos, tempPresentMove.GetLeftName), New Dot(tempPresentMove.GetEndPos, 
tempPresentMove.GetRightName), 2))
                moveIndex = tempGameState.Count - 1
                RemoveMove(tempAvailableMoves, tempGameState, moveIndex)
                If depth > 1 Then
                    score = Heuristic(tempPresentMove, tempGameState)
                End If

                If BoxesMade(tempPresentMove, tempGameState) >= 1 Then 'If 1+ box is made this player gets another turn; is added to cumulative 
score for this same depth
                    If score = 0 Then
                        score = Heuristic(tempPresentMove, tempGameState)
                    End If
                    score += Minimax(2, depth, tempGameState, tempAvailableMoves, tempPresentMove, bestMove, alpha, beta)
                Else score += Minimax(1, depth - 1, tempGameState, tempAvailableMoves, tempPresentMove, bestMove, alpha, beta)
                End If

                tempAvailableMoves.Insert(i, tempPresentMove)
                tempGameState.RemoveAt(moveIndex)

                If score > greatest Then
                    greatest = score
                    moveToReturn = tempPresentMove
                End If

                If greatest > alpha Then
                    alpha = greatest
                End If

                If beta <= alpha Then
                    Exit For
                End If

            Next
            If depth = AIdepth And currentMove Is Nothing Then 'returns ideal move out by reference on the uppermost depth (representing the next 
move in the real game)

                bestMove = moveToReturn

            End If

            Return greatest

        ElseIf player = 1 Then 'Minimising
            greatest = 999999

            For i = 0 To tempAvailableMoves.Count - 1
                score = 0
                tempPresentMove = tempAvailableMoves(i)
                tempGameState.Add(New Move(New Dot(tempPresentMove.GetStartPos, tempPresentMove.GetLeftName), New Dot(tempPresentMove.GetEndPos, 
tempPresentMove.GetRightName), 1))
                moveIndex = tempGameState.Count - 1
                RemoveMove(tempAvailableMoves, tempGameState, moveIndex)
                If depth > 1 Then
                    score = Heuristic(tempPresentMove, tempGameState)
                End If
                If BoxesMade(tempPresentMove, tempGameState) >= 1 Then
                    If score = 0 Then
                        score = Heuristic(tempPresentMove, tempGameState)
                    End If
                    score += Minimax(1, depth, tempGameState, tempAvailableMoves, tempPresentMove, bestMove, alpha, beta)
                Else score += Minimax(2, depth - 1, tempGameState, tempAvailableMoves, tempPresentMove, bestMove, alpha, beta)
                End If

                tempAvailableMoves.Insert(i, tempPresentMove) 'adds moves back to the available moves once one possible route has been searched
                tempGameState.RemoveAt(moveIndex) 'removes temporary moves made on the temporary board state once a route has been searched

                If score < greatest Then
                    greatest = score
                    moveToReturn = tempPresentMove
                End If

                If greatest < beta Then
                    beta = greatest
                End If
                If beta <= alpha Then
                    Exit For
                End If

            Next

            Return greatest

        End If

    End Function

    Private Sub ComputerMove(ByRef bestMove As Move)
        Dim temp As Move
        Minimax(2, AIdepth, lines, availableMoves, temp, bestMove, -999999, 999999) 'Initial minimax call. Will determine AI move, passing it out
of the minimax by reference with parameter bestMove
    End Sub

    Private Sub btn_Back_Click(sender As Object, e As EventArgs) Handles btn_Back.Click
        Hide()
        DifficultySelect.Show()
    End Sub

    Private Sub btn_Exit_Click(sender As Object, e As EventArgs) Handles btn_Exit.Click
        Application.Exit()
    End Sub
End Class

Class Dot
    Private xCoord As Integer
    Private yCoord As Integer
    Private xyCoords As Point
    Private id As String

    Public Sub New(ByVal location As Point, ByVal DotID As String)
        xCoord = location.X
        yCoord = location.Y
        xyCoords = location
        id = DotID
    End Sub

    Public Function GetX()
        Return xCoord
    End Function

    Public Function GetID()
        Return id
    End Function

    Public Function GetY()
        Return yCoord
    End Function

    Public Function GetXY()
        Return xyCoords
    End Function

End Class

Class Move
    Private firstXY As Point
    Private lastXY As Point
    Private name As String
    Private player As Integer

    Public Sub New(ByVal firstDot As Dot, ByVal lastDot As Dot, whichPlayer As Integer)
        firstXY = firstDot.GetXY
        lastXY = lastDot.GetXY
        name = firstDot.GetID & "-" & lastDot.GetID
        player = whichPlayer

    End Sub

    Public Function GetLineColour()
        If player = 1 Then
            GetLineColour = Color.Red
        ElseIf player = 2 Then
            GetLineColour = Color.DodgerBlue
        End If
    End Function

    Public Function GetStartPos()
        Return firstXY
    End Function

    Public Function GetEndPos()
```

```
        Return lastXY
    End Function

    Function GetLineName()
        Return name
    End Function

    Public Function GetLeftName()
        Return CInt(name.Split("-")(0))
    End Function

    Public Function GetRightName()
        Return CInt(name.Split("-")(1))
    End Function

    Public Function GetOrientation()
        If GetRightName() = GetLeftName() + 1 Then
            Return "V"
        Else Return "H"
        End If
    End Function

End Class


Class Box
    Private boxRoofStartCoords As Point
    Private boxRoofEndCoords As Point
    Private player As Integer
    Private numBoxesToLabel As Integer
    Private labelCoords As Point

    Public Sub New(ByVal upperLineStartCoords As Point, ByVal upperLineEndCoords As Point, ByVal whoseTurn As Integer, ByVal howManyBoxes As
Integer)
        boxRoofStartCoords = upperLineStartCoords
        boxRoofEndCoords = upperLineEndCoords
        player = whoseTurn
        numBoxesToLabel = howManyBoxes
    End Sub

    Public Function SetLabelPosition()
        labelCoords.X = (boxRoofStartCoords.X + boxRoofEndCoords.X) / 2
        labelCoords.Y = boxRoofStartCoords.Y + (labelCoords.X - boxRoofStartCoords.X)
        Return labelCoords
    End Function

    Public Function GetLabelTextColour()
        If player = 1 Then
            GetLabelTextColour = Color.Red
        ElseIf player = 2 Then
            GetLabelTextColour = Color.Blue
        End If
    End Function

    Public Function GetLabelText() As String
        If player = 1 Then
            GetLabelText = "P1"
        ElseIf player = 2 Then
            GetLabelText = "P2"
        End If
    End Function

End Class
```

AI Stats Form (Handling Display of AI Data)

This form is used to review total AI scores. It makes use of a listbox to store headings and values and pulls the values from the AIWins structure in the main form (which takes/stores data in CSV). It additionally calculates the total win rate using the total wins and total games for each difficulty level.

```
Public Class AIStats
    Private strFormat As String = "{0, -14}{1, -14}{2, -14}{3, -14}" 'String format that lays out the headings and values clearly

    Private Sub AIStats_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        UpdateListbox()
    End Sub

    Private Sub UpdateItems()
        Dim winRate As String

        For row = 0 To 2
            winRate = CStr(CalcWinRate(Game.AIWins(row).totalWins, Game.AIWins(row).totalGames)) & "%"
            lst_Stats.Items.Add(String.Format(strFormat, Game.AIWins(row).difficulty, Game.AIWins(row).totalWins, Game.AIWins(row).totalGames, 
winRate))
        Next
    End Sub

    Private Function CalcWinRate(ByVal numWins As Integer, ByVal numGames As Integer)
        Dim rate As Decimal
        Dim final As Integer
        If numWins = 0 Then
            final = 0
        Else
            rate = (numWins / numGames) * 100
            final = Math.Round(rate, 0)
        End If
        Return final
    End Function

    Private Sub btn_Back_Click(sender As Object, e As EventArgs) Handles btn_Back.Click
        Hide()
        DifficultySelect.Show()
    End Sub

    Private Sub UpdateListbox()
        lst_Stats.Items.Add(String.Format(strFormat, "Difficulty", "Total Wins", "Total Games", "Win Rate"))
        UpdateItems()
    End Sub
    Private Sub btn_Refresh_Click(sender As Object, e As EventArgs) Handles btn_Refresh.Click
        lst_Stats.Items.Clear()
        UpdateListbox()
    End Sub

    Private Sub IfClosed(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing 'Completely closes
program if X is clicked and asks user to confirm
        DifficultySelect.Show()
    End Sub
End Class
```

Difficulty Select Form (Startup Form)

This form is the first form the user is presented with upon loading the program. The buttons correlate to the number of depth levels the minimax will use as the maximum depth and sets this in the main form. It also prepares the correct grid size using a combo box, presenting a 4x4, 5x5 or 6x6 grid to the user back in the main form depending on the option selected by the user.

```
Public Class DifficultySelect
    Public AILevels As Integer
    Public gridDimension As Integer

    Private Sub btn_Easy_Click(sender As Object, e As EventArgs) Handles btn_Easy.Click
        AILevels = 1
        GetGridSize()
        If cmb_GridSize.Items.Contains(cmb_GridSize.Text) = True Then
            Game.SetNewGame()
            Hide()
            Game.Show()
        Else
            MsgBox("Please select a valid grid size")
        End If
    End Sub

    Private Sub btn_Medium_Click(sender As Object, e As EventArgs) Handles btn_Medium.Click
        AILevels = 3
        GetGridSize()
        If cmb_GridSize.Items.Contains(cmb_GridSize.Text) = True Then
            Game.SetNewGame()
            Hide()
            Game.Show()
        Else
```

```
            MsgBox("Please select a valid grid size")
        End If
    End Sub

    Private Sub btn_Hard_Click(sender As Object, e As EventArgs) Handles btn_Hard.Click
        AILevels = 5
        GetGridSize()
        If cmb_GridSize.Items.Contains(cmb_GridSize.Text) = True Then
            Game.SetNewGame()
            Hide()
            Game.Show()
        Else
            MsgBox("Please select a valid grid size")
        End If
    End Sub

    Private Sub btn_ViewAIScores_Click(sender As Object, e As EventArgs) Handles btn_ViewAIScores.Click
        Hide()
        AIStats.Show()

    End Sub

    Private Sub IfClosed(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing 'Completely closes
program if X is clicked and asks user to confirm
        Application.Exit()
    End Sub

    Private Sub btn_Exit_Click(sender As Object, e As EventArgs) Handles btn_Exit.Click
        Application.Exit()
    End Sub

    Private Sub GetGridSize()
        If cmb_GridSize.SelectedItem = "4x4" Then
            gridDimension = 4
        ElseIf cmb_GridSize.SelectedItem = "5x5" Then
            gridDimension = 5
        ElseIf cmb_GridSize.SelectedItem = "6x6" Then
            gridDimension = 6
        End If
    End Sub

End Class
```