Using Artificial Intelligence techniques to design an algorithm that can play Noughts and Crosses

Henry P Senior

2016

Abstract

Noughts and Crosses is a game that many people learn as young children. This project researches the different methods of creating a computer programme that is able to play Noughts and Crosses against a human player, with the computer winning or drawing every game. Of the algorithms researched, the Minimax Algorithm was best suited for use within my programme.

1 Definitions

- Artificial Intelligence (AI) - A field of Computer Science that focuses on the understanding and creation of intelligent agents.
- Algorithm - A set of instructions that achieves a specific task.
- Child Node - A node that extends from another node. It is said to be the child of the node it extends from.
- Code Library - Sections of pre-written code that programmers can use to complete specific tasks such as writing data to files.
- Node - A point in a network or diagram at which lines or pathways intersect or branch.
- Programming Paradigm - The style of programming a particular programming language.
- Search Space - The feasible region defining the set of all possible solutions.
- Subroutine - A section of code that is given a unique name. When that name is called within the code, the section of code assigned to the name is executed.
- Syntax - The grammatical rules a programming language must follow.
- User Interface (UI) - The way the user will interact with my programme.

2 What is Artificial Intelligence?

Artificial Intelligence (AI) is the term given to the field of Computer Science that focuses on both understanding and building intelligence into systems. The term was first used by J McArthy *et al* in their 1955 paper, *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence* [8]. AI draws on a wide range of different subjects, from Psychology to Pure Mathematics. AI is a fast paced field that effects all of us.

3 Possible Projects

Knowing that I wanted to create an artefact that could simulate a basic level of intelligence I needed to research a suitable project. Looking through *Artificial Intelligence A Modern Approach* and *Artificial Intelligence Structures and Strategies for Complex Problem Solving* I found two possible projects that I could undertake. Both projects were described as good examples for implementing basic AI. The projects were; *"A Noughts and Crosses Game"*, or a *"*3×3 *Sliding Tile Puzzle Solver"*. Due to the complex logic behind the sliding puzzle solver, it would be more feasible to make a programme that can play Noughts and Crosses.

4 Project Overview

Focusing my research around creating a basic AI programme that can play Noughts and Crosses against a human player without losing, the main challenge is to ensure that the programme will always play a perfect game of Noughts and Crosses. Thus the programme will always win or draw.

5 Noughts and Crosses

Noughts and Crosses is a two player game that takes place on a 3×3 grid. Each player is assigned either *"O"* or *"X"* and takes turns placing their symbol on the grid. The aim of the game is to be the first player to get 3 in a row. If both people play a "perfect" game, then the game will end with a draw. A win will only occur if one of the players has played an imperfect game.

Even though Noughts and Crosses is a simplistic game, there are a surprising number of different possible games. In total, there are $255,168$ individual games.[2]

6 Programming Languages and Programming Paradigms

The best way to think about programming languages is to think of them as tools that allow programmers to give instructions to a computer. The amount of programming languages is unknown, as new ones are constantly released. The best estimate to the amount of languages available is the GitHub language identifier; which can currently identify 371 programming languages (August 2015) [5].

If programming languages are like tools, then they come in many forms. The three main forms (paradigms) that programming languages come in are; Imperative, Object Oriented and Functional.

6.1 Programming Paradigms

As a programming language is the tool I will be using for the creation of my artefact, it is important to know which type of tool to use. This is why I decided to research programming paradigms. The research will help me better understand how to use the programming language I chose to use as well as give me a wider appreciation for the different types of programming languages.

6.1.1 Imperative Programming

> "With an imperative approach, a developer writes code that describes in exacting detail the steps that the computer must take to accomplish the goal."[9]

Imperative Programming is the oldest programming paradigm. It has a simple approach that makes it good for solving small problems. Imperative programming treats software development like flat pack furniture; everything follows a chronological order. Programmers use subroutines to split code into different parts. These parts then come together to make the final piece of software.

6.1.2 Object Oriented Programming

An Object Oriented language is more appropriate for larger applications than imperative languages. A programme is split into objects which represent different parts of the programme. Each object then contains the subroutines specific to that object.

If a car was a programme, then there would be objects for the wheels, engine, seats etc. All the components that make up the car would have their own object.

6.1.3 Functional Programming

> "The functional programming paradigm was explicitly created to support a pure functional approach to problem solving."[9]

When programming in a functional language, operations are treated as mathematical functions. They are good at solving mathematical problems and inherit features of mathematics such as immutable data types. A variable in maths is set to a value and then it cannot be changed. Variables are treated exactly the same in functional programming languages, unlike in the other two styles where data is mutable.

6.2 Programming Languages

As there are a wide range of programming languages, it is important to pick one that is appropriate for the project. If I were to pick an unsuitable programming language then the project would become significantly more difficult. It is important to pick the right tool, so researching programming languages was important.

6.2.1 F#

F# is a programming language created by Microsoft Research. The aim of the project was the make a Functional Programming Language that allows developers to write simple code that is capable of solving complex problems. It also aims to reduce the time taken between the writing of code and that code's deployment. F# would be a suitable programming language as my project has a strong mathematical basis.[12] [4]

6.2.2 C++

C++ is a programming language created by Bjarne Stroustrup and was originally designed to work alongside the older C programming language. The name "C++" comes from some old programming short hand. If a programmer wants to +1 to a variable they can type: $VariableName++$. As C++ was designed to improve on C, it was aptly named C++.

C++ shares lots of common features with the C programming language, such as syntax. However, whilst C is an imperative language, C++ is Object Oriented. This is by far the biggest difference between the two languages and one of the ways C++ is more powerful than the original C language. [14]

6.2.3 C#

C# is a programming language created by Microsoft and released in 2000. It aims to be a modern programming language that uses the Object Oriented programming paradigm. However, as C# has grown, it has incorporated features of the Functional programming paradigm such as immutable data types. The syntax, or grammar, of C# was intentionally designed to be similar to other popular programming languages such as C, C++ and Java this allows for developers to easily pick up C#. The high standards that are used at Microsoft to maintain the language are shown by the fact it is standardised by both ECMA and ISO/IEC, the two main bodies that maintain standards within the Computing industry. [10] [12]

7 Ways of Playing Noughts and Crosses

7.1 Random Position Generation

This method involves generating a number at random which corresponds with a certain position on the game board. If the position is free then the AI will play there, if not, it will carry on generating numbers until it can play. This approach is highly simplistic and cannot replicate intelligence. It would be obvious what the computer would be doing as it would play moves of little to no strategic value.

This method could be improved by adding instructions that tell the programme to make certain moves if a specific case occurs. For example, it could be programmed to block the player if they have 2 tiles in a row. The programme however, would still fail to replicate intelligence.

7.2 Search

Search is one of the original forms of AI. It involves the computer searching through all the possible scenarios that could possibly occur and then going down the best route. In the context of Noughts and Crosses, the computer will calculate every game that could ever be played and then attempt to use the path that uses the shortest amount of moves to reach a desired outcome.

Whilst search is a very primitive form of Artificial Intelligence, it can replicate intelligence when tasked with doing something trivial such as Noughts and Crosses. By using search, the computer can calculate all the possible states the game can be in. This allows the computer to tailor the game in its favour and theoretically play a perfect game.

If a search algorithm is deployed, then an accompanying Path Finding Algorithm must also be used to find the quickest route through the tree generated by the Search Algorithm. A popular Path Finding Algorithm is Dijkstra's Shortest Path Algorithm.

7.2.1 Breadth First Search

The Breadth First Search (BFS) algorithm is a basic search algorithm. It is named as such due to its focus on the breadth of a search rather than the depth of a search. The algorithm will search all adjacent nodes before descending down a branch. It will loop through until all the nodes in the graph have been searched. [6]

One application of the BFS algorithm is network mapping. It is used to search computer networks, showing the connections between servers. The diagram below shows a network map of the early ARPANET (Advanced Research Projects Agency Network)starting from node MIT. The ARPANET was an early research project that later developed into what we now call the Internet.

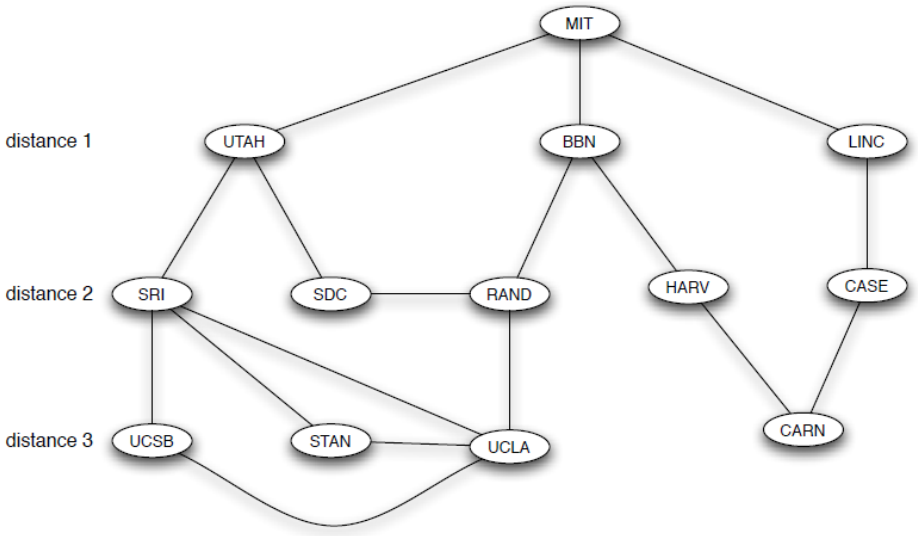


Figure 1: Breadth First Search of the Early ARPANET

Search algorithms encounter problems when faced with a large search space. The larger the search space, the more memory the algorithms need. The table below shows how more time and memory is needed for larger search spaces. [13]

Depth	Nodes	Time	Memory
2	1100	0.11 Seconds	1 Megabyte
4	111,100	11 Seconds	106 Megabytes
6	10^7	19 Minutes	10 Gigabytes
8	10^9	31 Hours	1 Terabyte
10	10^{11}	129 Days	101 Terabytes
12	10^{13}	35 Years	10 Petabytes
14	10^{15}	3523 Years	1 Exabyte

Table 1: Memory Usage of Breadth First Search

A Note on file sizes: 100 *Megabytes is about the size of a photo,* 8 *Gigabytes is the size of an average USB flash drive, the average hard drive size on a desktop computer is* 1 *Terabyte.* 1 *Petabyte is* 1024 *Terrabytes whilst 1 Exabyte is* 1024 *Petabytes - this means that a BFS searh of* 10^{15} *Nodes requires the space of* 1048576 *Desktop hard drives!*

7.2.2 Depth First Search

The Depth First Search (DFS) algorithm approaches search slightly differently; searching child nodes first. The algorithm will search until it reaches the deepest level of the tree. Then the algorithm will "reverse" until it reaches the next shallowest unexplored node. Just as the BFS, the algorithm will loop until all the nodes have been read. [6]

A large advantage the DFS algorithm has over the BFS is the amount of memory it uses. Compared with the BFS algorithm, it uses much less memory. The memory usage of the DFS algorithm is given by: $bm + 1 nodes$ [6]. Where b is the branching factor and m is the maximum depth. To compare the memory uses of the BFS and DFS see the below table:

	Depth First Search	Breadth First Search
Depth	12	12
Branching Factor	10^{13}	10^{13}
Memory Usage	$118 Kilobytes$	$10 Petabytes$

Table 2: Memory Usage of Depth First Search Compared to Breadth First Search

A Note on memory usage: *A large text document takes up around* 100 *kilobytes*

7.2.3 Dijkstra's Algorithm

After a search algorithm has been executed, a path finding algorithm can be used on the graph that is built up by the search. The algorithm will traverse through the tree, finding the shortest path between two nodes (start and end). One popular Path Finding Algorithm is Dijkstra's Algorithm for The Shortest Route.

First proposed by Edsger W Dijkstra in his 1959 paper *A Note on Two Problems in Connexion with Graphs.*

Dijkstra's Algorithm takes a start node and an end node. It then maps which nodes connect together, keeping track of the distance travelled. The algorithm will loop until the end node is reached. It will then return a list of node names, showing the list of nodes that make up the shortest route.

In Dijkstra's Algorithm, the work done at each step is approximately proportional to the number of branches that have not been permanently labelled. This means the total work is possibly: $\frac{1}{2}n(n-1)$ where n is the number of branches. Dijkstra's Algorithm therefore has quadratic complexity so doubling the amount of branches leads to about four times the effort. [3]

7.3 Game Theory Algorithm

Game Theory is a field of Mathematics that focuses on the study of "games". Whilst the term "game" can be used to refer to real games such as Chess or Noughts and Crosses, Game Theory uses the term to mean any scenario where there are strategies to adopt and outcomes or prizes to win. The theory focuses what happens when these "games" are played rationally. Can we predict the outcome of the game? How can we predict the outcome? Algorithms such as these can be applied to Noughts and Crosses as one player aims to maximise the strategic value of their move whilst also minimising the effectiveness of their opponent's next move.

7.3.1 Minimax Algorithm

The Minimax algorithm was one of the earliest algorithms to come out of Game Theory. Proposed by John Von-Neumann, the *Maximin Principle* focuses on Zero Sum games(games where the gain of one player is equalised by the loss of the opponent). The principle aims to "maximise" the effectiveness of a move at the expense of "minimising" the effectiveness of the opponent's. [15]

Von-Neumann proposed an algorithm that was capable of applying the *Maximin Principle* to games. This algorithm was known as the *Minimax Algorithm*

The Minimax Algorithm is named thus as it tries to *minimise* the potential loss, even in a worse case scenario, whilst trying to *maximise* the possible gain (helping achieve the equilibrium of the *Maximin Principle*). The algorithm searches through possible moves that can be played and scores them. The higher the score, the higher chance that move will lead to a win. It will then play the move that has the highest score. [7]

The algorithm runs through all the possible moves and creates a tree, similar to that created in the BFS or DFS search algorithms. However, the generated tree doesn't hold every possible game that could be played, it is instead limited to a certain depth. The moves held in this tree are then scored according to the *Maximin Principle* and the algorithm will return the position that it scores as having the highest strategic value.

8 How the Computer Plays Noughts and Crosses

In part, the programming language you chose dictates the paradigm you must programme in. However, some languages allow for multiple paradigms to be used.

I have decided to use C# as the programming language for my EPQ. This is due to its similarity to Visual Basic (the language used in my A-level Computing). Another advantage of C# is that it allows for multiple paradigms to be used - allowing me to tailor my choice of paradigm to the section I am programming.

Although C++ is a powerful language, I am familiar with the structure of C# programmes as they are in the same format as Visual Basic programmes. As I am using C# I will be using the Object Oriented programming paradigm as it is the best paradigm supported by the language.

Having never used a Functional Programming Language before, the style change in programming techniques would have been substantial. The style change would have resulted in me having the change the way I think about programming and the way I structure my programmes as well as learning a new programming language. As a result, it would be unfeasible to use F# to create a successful project within the given time scale.

To ensure that my programme always plays a perfect game of Noughts and Crosses I will use the Game Theory Algorithm. Although I could use a Search Algorithm to successfully complete my project, the programme would be very slow and use lots of memory on the computer it is running on. A search algorithm would also increase the complexity of the programme as a path finding algorithm would also have to be deployed. This would result in my programme using more memory and it taking longer for a move to be calculated. A Game Theory Algorithm such as the Minimax Algorithm would be more suitable for my programme as it uses less memory and has a quicker computation time. As Noughts and Crosses is a Zero Sum game, it is possible for the Minimax Algorithm to be used.

9 Real World Applications

9.1 Programming Languages

Computers are a fundamental part to modern life. Most things are controlled by computers, from controlling a washing machine to multi-million dollar trades on Wall Street. Every programme has to be written in a programming language, meaning that most programming languages have a crucial role in today's society.

9.2 AI

The use of AI is more widely spread than people originally think; being used by Amazon to suggest items a customer may wish to purchase. Another popular use of AI is in smart phones. Google, Apple and Microsoft all have virtual assistants built into their phones that help people book appointments, call friends or search for something online.

9.3 Search Algorithms

Search Algorithms are commonly used alongside Path Finding Algorithms in navigation systems. A tree is made of possible routes that could be taken to get from point A to point B. The Path Finding Algorithm then finds the quickest route along the tree.

9.4 Game Theory

Game Theory was originally dismissed as another useless theoretical field of mathematics. That changed when it was applied to the auctioning of radio frequencies used for mobile phones. The American Taxpayer made a profit of $20 Billion. This proved that Game Theory did in fact have real world applications. Game Theory is now widely used in the stock market and is used in times of war to help Governments form strategies. [1]

References

[1] Binmore, K. 2007. Game Theory a Very Short Introduction. 2007. Oxford. Oxford University Press

[2] Bottomly, H. How many Tic-Tac-Toe (noughts and crosses) games are possible?. Retrieved Jan 2016. http://www.se16.info/hgb/tictactoe.htm

[3] Dijkstra, E. 1959. A note on two problems in connexion with graphs. Numer. Math. Numerische Mathematik (1959), 269271.

[4] F# Software Foundation. About F#. Retrieved Feb 2016 http://fsharp.org/about/index.html

[5] GitHub Inc 2011. github/linguist. (August 2011). Retrieved Aug 2015 from https://github.com/github/linguist

[6] Jones, T. 2005. AI application programming 2nd ed., Hingham, Mass.: Charles River Media.

[7] Luger, G. 2005. Artificial intelligence: structures and strategies for complex problem solving 5th ed., Harlow, England: Addison-Wesley.

[8] McCarthy et al. A PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE. Retrieved Aug 2015. Www-formal.stanford.edu

[9] Microsoft. Functional Programming vs. Imperative Programming. Retrieved Aug 2015 from https://msdn.microsoft.com/en-gb/library/bb669144.aspx

[10] Microsoft. 2013. C# Language Specification Version 5.0 1st ed.

[11] Parramore, K. 2004. Decision mathematics 1 3rd ed., London: Hodder and Stoughton.

[12] Petricek, T and Skeet, J. Overview: Introducing F# and Functional Programming. (2010). Retrieved 2015 from https://msdn.microsoft.com/library/hh297121(v=vs.100).aspx

[13] Russell, S and Norvig, P. 2003. Artificial intelligence: a modern approach 2nd ed., Upper Saddle River, N.J.: Prentice Hall/Pearson Education.

[14] Stroustrup, B. 1991. The C++ Programming Language. Second Edition. New Jersey. Addison-Wesley.

[15] Von-Neumann, J. Zur Theorie der Gesellschaftsspiele (In German). Math. Ann. 100 (1928) 295-320

A Figures

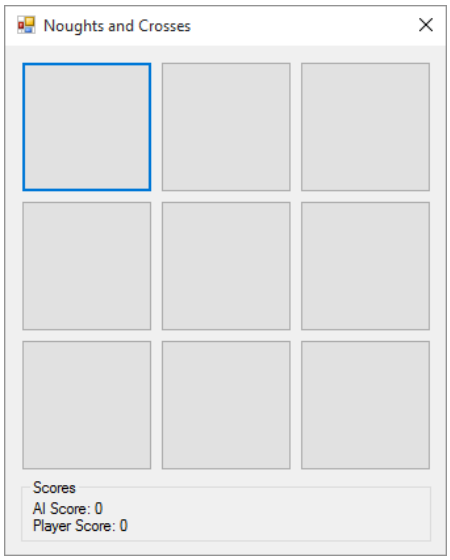
7.2.1 : Breadth First Search of the Early ARPANET

B Tables

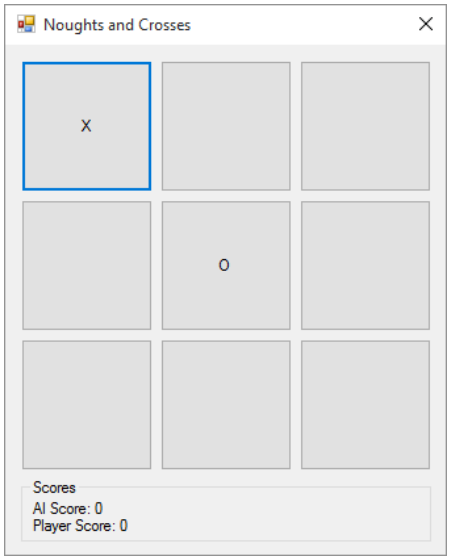
7.2.1 : Memory Usage of Breadth First Search
7.2.2 : Memory Usage of Depth First Search Compared to Breadth First Search

C Artefact Screenshots

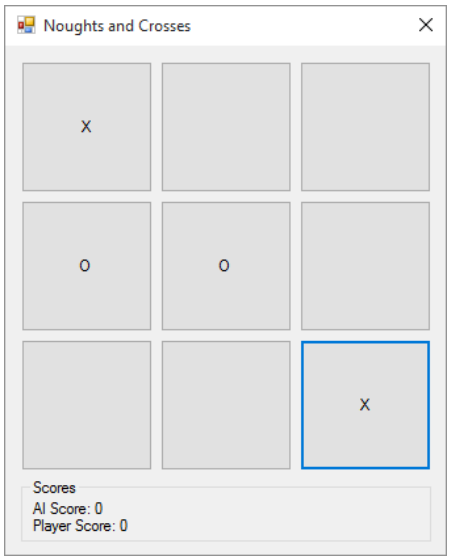
C.1 Draw



Programme loads up and allows player to make the first move.



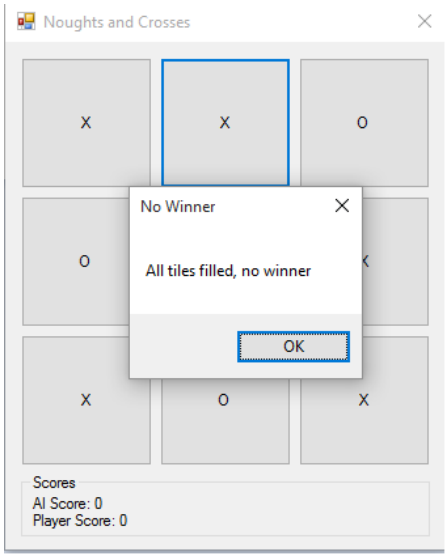
Once the player have made their first move, the AI plays almost instantly.



The Player makes their second move, as does the AI.

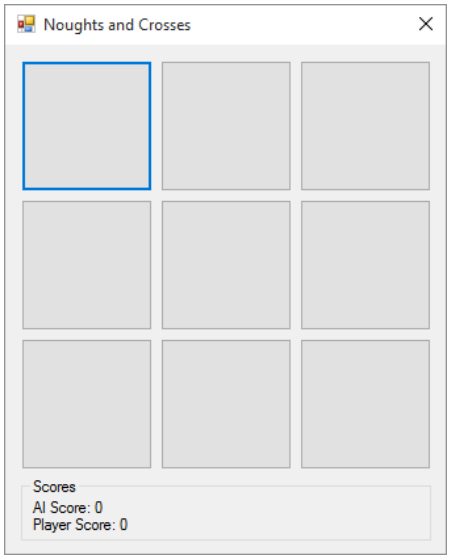
The Player makes their third move, as does the AI.

The Player makes their fourth move, as does the AI.

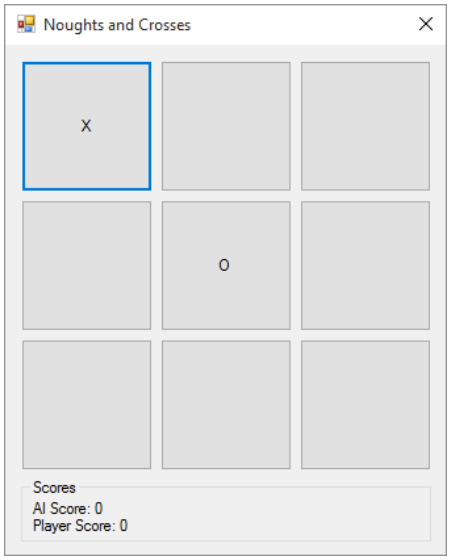


The player fills the final space, ending the game on a draw.

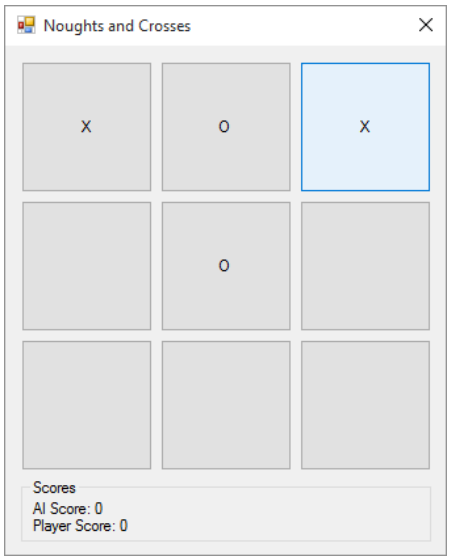
C.2 AI Wins



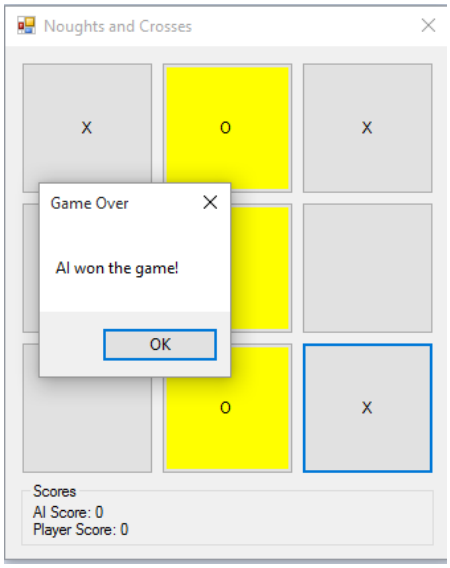
Programme loads up and allows player to make the first move.



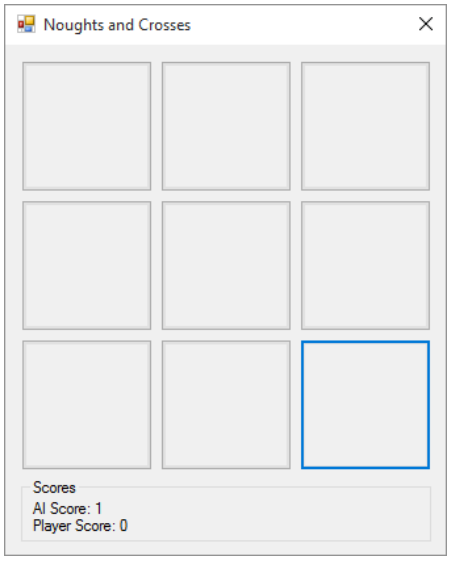
Once the player have made their first move, the AI plays almost instantly.



The Player makes their second move, as does the AI.



The Player fails to block the AI's attempt for 3 in a row. The AI calculates the winning move and plays it, resulting in a win. The programme highlights the 3 in a row and alerts the user to their loss.



The programme then resets and increases the winner's score.

D Code

D.1 Form1

```
using EPQ.Class_Files;
using System;
using System.Windows.Forms;

namespace EPQ
{
    public partial class Form1 : Form
    {
        private board b = new board();
        public search s = new search();

        private bool player = true;  //true = player     |     false = AI

        private int scoreX = 0;
        private int scoreO = 0;
        private int movesPlayed = 0;

        public Form1()
        {
            InitializeComponent();
        }

        //Load Form event
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        //update board
        private void update(int x, int y, Button btn)
        {
            //update player, competitor and button text if that position is free

            if (b.spaces[x, y] == competitor.freeSpace)
            {
                if (player == true)
                {
                    btn.Text = "X";
                    b.spaces[x, y] = competitor.player;
                    player = false;
                }
                else
                {
                    btn.Text = "O";
                    b.spaces[x, y] = competitor.AI;
                    player = true;
                }

                //incriment moves
                movesPlayed++;

                //check for 3 in a row
                checkForThree();
            }
            DataOut();
```

```
}

//Call AI move
private void AIMove()
{
    position bestMove = s.minimax(b, competitor.AI);
    update(bestMove.X, bestMove.Y, getBtn(bestMove.X, bestMove.Y));
}

private Button getBtn(int X, int Y)
{
    switch (Y)
    {
        case 0:
            if (X == 0) return btn1;
            if (X == 1) return btn2;
            if (X == 2) return btn3;
            break;

        case 1:
            if (X == 0) return btn4;
            if (X == 1) return btn5;
            if (X == 2) return btn6;
            break;

        case 2:
            if (X == 0) return btn7;
            if (X == 1) return btn8;
            if (X == 2) return btn9;
            break;

        //error handling
        default:
            MessageBox.Show("An error occured calculating the button pressed
            break;
    }
    return btn9;
}

//check 3 in a row
private void checkForThree()
{
    //check row
    checkWinner(btn1, btn2, btn3);
    checkWinner(btn4, btn5, btn6);
    checkWinner(btn7, btn8, btn9);

    //check column
    checkWinner(btn1, btn4, btn7);
    checkWinner(btn2, btn5, btn8);
    checkWinner(btn3, btn6, btn9);

    //check diagonally
    checkWinner(btn1, btn5, btn9);
    checkWinner(btn3, btn5, btn7);

    //check if all filled an no winner
    if (movesPlayed == 9)
```

```
            checkAllFilled();
        }

        //Check for winner
        private void checkWinner(Button btnOne, Button btnTwo, Button btnThree)
        {
            //if all three buttons have the same text and aren't blank
            if (btnOne.Text.Length + btnTwo.Text.Length + btnThree.Text.Length > 0)
            {
                if ((btnOne.Text == btnTwo.Text) && (btnTwo.Text == btnThree.Text))
                {
                    //set the colours the to indentify win
                    btnOne.BackColor = System.Drawing.Color.Yellow;
                    btnTwo.BackColor = System.Drawing.Color.Yellow;
                    btnThree.BackColor = System.Drawing.Color.Yellow;

                    //announce win
                    displayWinner(btnOne);

                    //reset board
                    resetBoard();
                }
            }
        }

        //check for all tiles filled but no winner
        private void checkAllFilled()
        {
            MessageBox.Show("All tiles filled, no winner", "No Winner");
            resetBoard();
        }

        //display winner and update the score
        private void displayWinner(Button btnWinner)
        {
            if (btnWinner.Text == "X")
                scoreX++;
            else
                scoreO++;

            if (btnWinner.Text == "O")
                MessageBox.Show("AI won the game!", "Game Over");
            else
                MessageBox.Show("Player won the game!", "Game Over");

            updateScore();
        }

        //update score board
        private void updateScore()
        {
            //update AI
            LblAI.Text = "AI Score: " + scoreO.ToString();

            //update Player
            LblPlayer.Text = "Player Score: " + scoreX.ToString();
        }
```

```
//reset board
private void resetBoard()
{
    //reset buttons
    btn1.Text = "";
    btn1.BackColor = DefaultBackColor;
    btn2.Text = "";
    btn2.BackColor = DefaultBackColor;
    btn3.Text = "";
    btn3.BackColor = DefaultBackColor;
    btn4.Text = "";
    btn4.BackColor = DefaultBackColor;
    btn5.Text = "";
    btn5.BackColor = DefaultBackColor;
    btn6.Text = "";
    btn6.BackColor = DefaultBackColor;
    btn7.Text = "";
    btn7.BackColor = DefaultBackColor;
    btn8.Text = "";
    btn8.BackColor = DefaultBackColor;
    btn9.Text = "";
    btn9.BackColor = DefaultBackColor;

    //reset board
    b = new board();

    //reset moves
    movesPlayed = 0;
}

//handle input//
private void btn1_Click(object sender, EventArgs e)
{
    if (b.spaces[0, 0] == competitor.freeSpace)
    {
        update(0, 0, btn1);
        AIMove();
    }
}

private void btn2_Click(object sender, EventArgs e)
{
    if (b.spaces[1, 0] == competitor.freeSpace)
    {
        update(1, 0, btn2);
        AIMove();
    }
}

private void btn3_Click(object sender, EventArgs e)
{
    if (b.spaces[2, 0] == competitor.freeSpace)
    {
        update(2, 0, btn3);
        AIMove();
    }
}
```

```
private void btn4_Click(object sender, EventArgs e)
{
    if (b.spaces[0, 1] == competitor.freeSpace)
    {
        update(0, 1, btn4);
        AIMove();
    }
}

private void btn5_Click(object sender, EventArgs e)
{
    if (b.spaces[1, 1] == competitor.freeSpace)
    {
        update(1, 1, btn5);
        AIMove();
    }
}

private void btn6_Click(object sender, EventArgs e)
{
    if (b.spaces[2, 1] == competitor.freeSpace)
    {
        update(2, 1, btn6);
        AIMove();
    }
}

private void btn7_Click(object sender, EventArgs e)
{
    if (b.spaces[0, 2] == competitor.freeSpace)
    {
        update(0, 2, btn7);
        AIMove();
    }
}

private void btn8_Click(object sender, EventArgs e)
{
    if (b.spaces[1, 2] == competitor.freeSpace)
    {
        update(1, 2, btn8);
        AIMove();
    }
}

private void btn9_Click(object sender, EventArgs e)
{
    if (b.spaces[2, 2] == competitor.freeSpace)
    {
        update(2, 2, btn9);
        AIMove();
    }
}

//OUTPUT Data Structures to console for debugging
private void DataOut()
{
    Console.Write(b.spaces[0, 0] + " | ");
```

```
            Console.Write(b.spaces[1, 0] + " | ");
            Console.Write(b.spaces[2, 0] + Environment.NewLine);

            Console.Write(b.spaces[0, 1] + " | ");
            Console.Write(b.spaces[1, 1] + " | ");
            Console.Write(b.spaces[2, 1] + Environment.NewLine);

            Console.Write(b.spaces[0, 2] + " | ");
            Console.Write(b.spaces[1, 2] + " | ");
            Console.Write(b.spaces[2, 2] + Environment.NewLine);

            Console.WriteLine("==================");
        }
    }
}
```

D.2 Board Class

```
using System.Collections.Generic;

namespace EPQ.Class_Files
{
    //enum to hold player
    public enum competitor
    {
        player = 1,
        AI = -1,
        freeSpace = 0
    }

    public class board
    {
        // 2D array to hold the spaces on the board and what player is in that space
        public competitor[,] spaces;

        // Initalise a new game board's spaces
        public board()
        {
            spaces = new competitor[3, 3] { { 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 } };
        }

        // Used to get or set a value on the board (spaces)
        public competitor this[int x, int y]
        {
            get
            {
                return spaces[x, y];
            }
            set
            {
                spaces[x, y] = value;
            }
        }

        // returns whether or not all the spaces on the board are full
        public bool isFull
        {
            get
```

```
    {
        foreach (competitor comp in spaces)
        {
            if (comp == competitor.freeSpace) return false;
        }
        return true;
    }
}

// gets maximum size of board
public int size
{
    get
    {
        return 9;
    }
}

// Returns a list of all the open positions on the game board
public List<position> openPositions
{
    get
    {
        List<position> openPositions = new List<position>();
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (spaces[i, j] == competitor.freeSpace)
                {
                    openPositions.Add(new position(i, j));
                }
            }
        }

        return openPositions;
    }
}

// Clones the game board
public board clone()
{
    board b = new board();
    b.spaces = (competitor[,])this.spaces.Clone();
    return b;
}

// determines if the current board has a winner
public competitor winner
{
    get
    {
        int count = 0;

        //columns
        for (int x = 0; x < 3; x++)
        {
            count = 0;
```

```
                for (int y = 0; y < 3; y++)
                    count += (int)spaces[x, y];

                if (count == 3)
                    return competitor.player;
                if (count == -3)
                    return competitor.AI;
            }

            //rows
            for (int x = 0; x < 3; x++)
            {
                count = 0;

                for (int y = 0; y < 3; y++)
                    count += (int)spaces[y, x];

                if (count == 3)
                    return competitor.player;
                if (count == -3)
                    return competitor.AI;
            }

            //diagnols right to left
            count = 0;
            count += (int)spaces[0, 0];
            count += (int)spaces[1, 1];
            count += (int)spaces[2, 2];
            if (count == 3)
                return competitor.player;
            if (count == -3)
                return competitor.AI;

            //diagnols left to right
            count = 0;
            count += (int)spaces[0, 2];
            count += (int)spaces[1, 1];
            count += (int)spaces[2, 0];
            if (count == 3)
                return competitor.player;
            if (count == -3)
                return competitor.AI;

            return competitor.freeSpace;
        }
    }
}

// Describes a position on the board
public struct position
{
    public int X;
    public int Y;
    public int score;

    public position(int x, int y)
    {
```

```
            X = x;
            Y = y;
            score = 0;
        }
    }
}
```

D.3 Search Class

```
using System;
using System.Windows.Forms;
using System.Collections.Generic;

namespace EPQ.Class_Files
{
    public class search
    {
        public position minimax(board gb, competitor player)
        {
            //variables
            position? bestPos = null;
            List<position> openPositions = gb.openPositions;
            board newBoard;

            //Loop all open spaces searching for the best move
            for (int i = 0; i < openPositions.Count; i++)
            {
                newBoard = gb.clone();
                position newPos = openPositions[i];

                newBoard[newPos.X, newPos.Y] = player;

                if (newBoard.winner == competitor.freeSpace && newBoard.openPositions
                {
                    position tempMove = minimax(newBoard, ((competitor)(-(int)player
                    newPos.score = tempMove.score;
                }
                else
                {
                    //score moves
                    if (newBoard.winner == competitor.freeSpace)
                        newPos.score = 0;
                    else if (newBoard.winner == competitor.player)
                        newPos.score = -1;
                    else if (newBoard.winner == competitor.AI)
                        newPos.score = 1;
                }

                //check it is the best move
                if (bestPos == null ||
                   (player == competitor.player && newPos.score < ((position)bestPos
                   (player == competitor.AI && newPos.score > ((position)bestPos).sc
                {
                    bestPos = newPos;
                }
            }

            //return the best move
```

```
                return (position)bestPos;
            }
        }
    }
```