Programming Fundamentals
CO-102

Sem II - MTE Project

on

SMART BOT FOR TIC-TAC-TOE

Submitted by:

KESHAV NATH - 2K20/A9/47

VATSAL MITTAL - 2K20/A9/82

Under the supervision of

Ms. Priya Singh

Department of Computer Science

Delhi Technological University

DECLARATION

We, Keshav Nath (2K20/A9/47) and Vatsal Mittal (2K20/A9/82), hereby certify that the work, which is presented in the Project - **Smart Bot for Tic-Tac-Toe**, submitted to The Department of Computer Science, is an authentic record of our own, carried out under the supervision of Ms. Priya Singh.

The work presented in this project has not been submitted to and is not under consideration for the award for any other credits in a course/degree of this or any other Institute/University.

ACKNOWLEDGEMENT

We convey our sincere gratitude to Ms. Priya Singh for her continuous guidance and help. Without her kind support, the completion of this project would not have been possible.

ABSTRACT

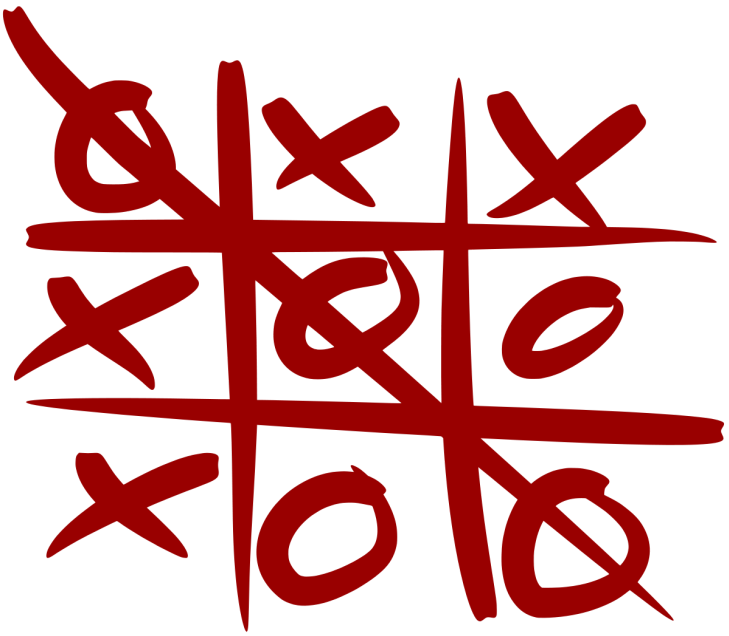
Tic-Tac-Toe is an old two-player game, played traditionally on paper. It involves a 3x3 Grid that can be filled with either an X or an O. Players select their token (X,O) and fill the grid turn-by-turn.

The first player to get 3 of their tokens in a row, column or diagonal, wins the game. It is convention for X to start the game.

Due to the nature of the simplicity of the game, there are a handful of strategies that prove most effective while playing the game. We aim to develop an automated bot for the game that plays at a considerably higher level than the average individual, thus proving to be a challenge to defeat.

The bot will be made in C Language with C Standard Libraries, incorporating functional programming concepts to ensure clarity and comprehensibility.

The ultimate objective is to make the bot completely undefeatable, irrespective of the situation it faces. Additionally, the bot should present the same level of skill with both tokens.



A standard game of TicTacToe

INDEX

INTRODUCTION 4
- **What is Tic-Tac-Toe?** 4
- **What is a bot?** 5
- **Project Objective** 5

ENVIRONMENT AND FRAMEWORK 6

BASIC LOGICAL FLOW 7

MODULAR BREAKDOWN 8
- **Globals** 8
- **Turn-By-Turn Approach** 8
- **Checking for Wins** 8
- **Checking for Losses** 9
- **Optimum Strategy** 9

BOT STRATEGY AS ‘O’ 10
- **Key Spots** 11
- **X takes Center** 11
- **X takes Corner** 11
- **Logical Flowchart** 12

BOT STRATEGY AS ‘X’ 13
- **Key Spots** 13
- **Starting from Center** 13
  - 1 - ‘O’ Takes Side Square 13
  - 2 - ‘O’ Takes a Corner 14
- **Starting from Corner** 14
  - 1 - ‘O’ Takes Side Square 14
  - 2 - ‘O’ Takes Center 14
- **Strategy Decision** 15
- **Logical Flowchart** 15

GAMEPLAY SAMPLE 16

MOTIVATIONS 20

CONCLUSIONS 20

FUTURE WORK 21

REFERENCES 22

INTRODUCTION

What is Tic-Tac-Toe?

Tic-Tac-Toe / Noughts and Crosses / Xs and Os is an age-old board game played by two people, competing against each other. It is commonly played with paper and pen.

A Tic-Tac-Toe board can be described as an empty 3X3 Grid. Players have to choose between ‘X’ and ‘O’ as their token or character. The grid is then filled turn-by-turn with X and O, with the person who selected the ‘X’ token starting first by filling ‘X’ in the empty grid.

The game is won when a player has filled either a row, a column or a diagonal with their token without interruption from the other player.

A draw is also a legal result and occurs when all the 9 grid-squares are filled without any token having achieved a winning condition.

The above diagram shows a standard game of TicTacToe with X winning on the bottom row.

The simplicity of the game, and the ability to play it anywhere with anyone, has made it historically popular, dating back to 1st Century B.C. in the Roman Empire, and it has been ever-present to this day.

Due to the relatively limited number of unique options, a handful of strategies for the game are considered to be the best and most likely to succeed, so they are the only ones that need to be involved to make a comprehensive bot.

What is a bot?

A bot can be described as a piece of software that runs autonomously (often based on scripts) in order to replace a human being (to the best of its ability) at a specific task.

Bots are most commonly used in menial, repetitive tasks, such as: sending mass emails, replying to FAQs, repairing grammatical errors in a document etc.

The word “bot” implies a level of simplicity, differentiating it from advanced machine learning-based Artificial Intelligence. While ML Models learn on their own from data while given some guidelines, Bots run on premade scripts defined by the creator to perform the exact task, with as many use cases as possible predicted and accounted for in the scripts.

Project Objective

The final objective of this project is to investigate the various strategies of Tic-Tac-Toe and create a bot that not only knows how to counter them, but also to implement them efficiently. This bot is unique because, while having the ability to defend itself and prevent a loss, it also knows how to play aggressively and win.

ENVIRONMENT AND FRAMEWORK

The bot is made entirely using only the **C Programming Language** and its native libraries, making it easily reproducible.

The C Programming Language was selected due to its high speed and procedural structure, allowing the ability to create a fast, responsive and optimizable program that can run on most windows-based devices.

The C Packages used are:

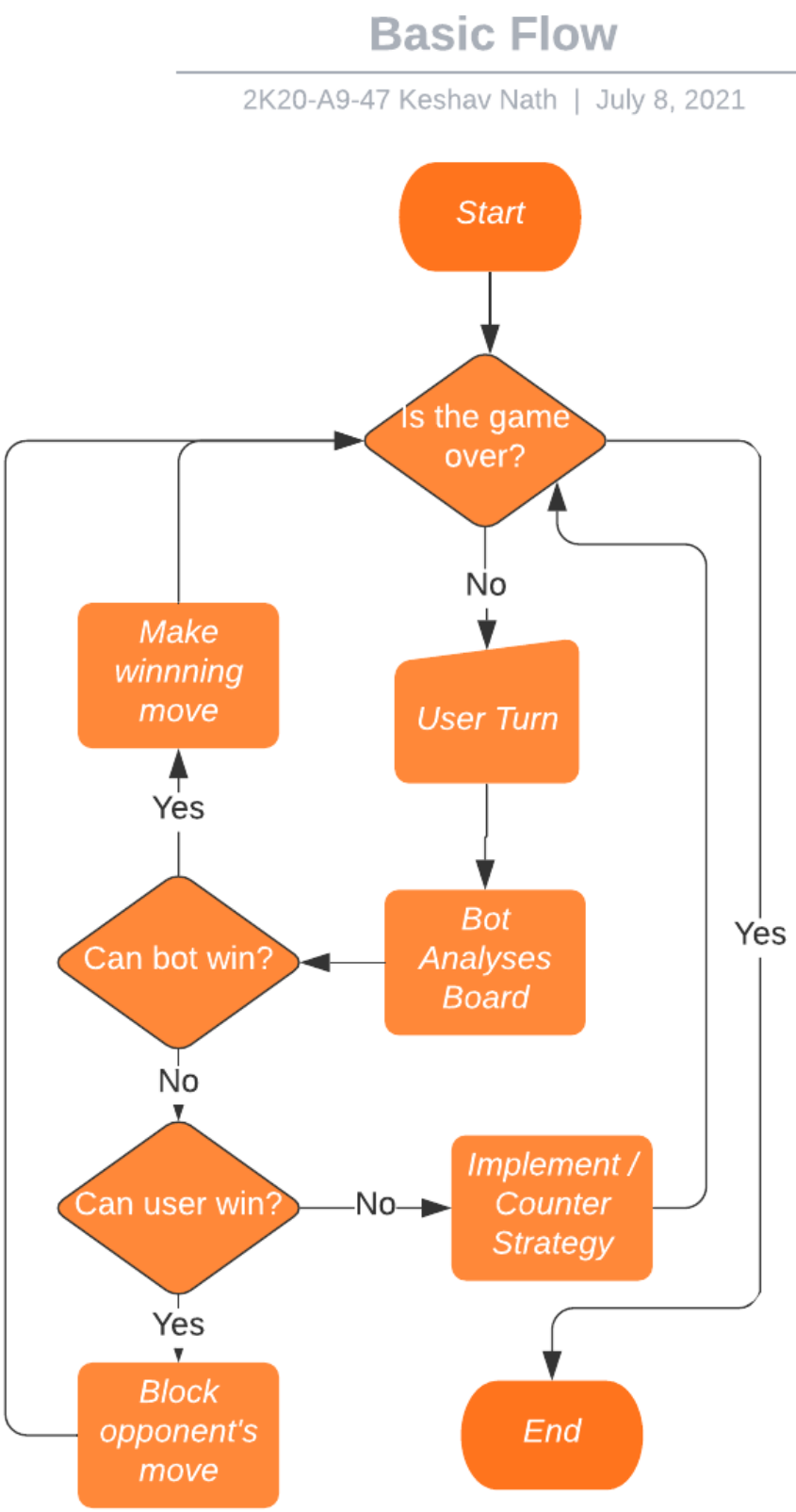
1. stdlib.h
2. stdio.h
3. conio.h
4. time.h

The creation and programming of the bot was done using the **Dev C++ IDE**, compiled using MinGW on Windows. The bot was tested comprehensively on various windows systems and on other compilers to ensure compatibility.

Dev C++ was used to create the program due to its simple yet comprehensive UI, and the fact that the MinGW Compiler comes pre-installed with it, eliminating compatibility issues, creating a well-stacked environment for C programming.

The bot is wrapped in an entire Tic-Tac-Toe program, providing an interactive gameplay experience to assess the features and decision-making of the bot. The entire application, in its current form, is a console application without any GUI.

BASIC LOGICAL FLOW



MODULAR BREAKDOWN

Globals

The Game Board is a 2D 3x3 Array initialized with ‘0’s. On the game board, 0 represents an empty slot, 1 represents ‘O’ and 2 represents ‘X’.
An integer ‘turns’ initialized to 0 is used to count the total number of turns that have taken place in the game.

Turn-By-Turn Approach

The game takes a turn-by-turn approach. Each turn follows a similar pattern.
For a user turn:

1) Board is displayed.
2) The player is asked for Row and Column to put their token.
3) If valid, the board is updated with the user’s token.
4) The game checks if the user has won the game, or if it's a draw.
5) If the game is not finished, the bot plays its turn.

For a Bot turn:

1) The bot checks if it can win the game in this turn. If it can, it makes that move.
2) If not, the bot checks if the user has any way to win the game. If it can, it blocks that move.
3) If not, the bot dives through its strategies and makes the optimal move.
4) The game checks if the bot has won the game, or if it's a draw.
5) If the game is not finished, the user plays their turn.

Checking for Wins

The first priority for the bot is checking if it can win the game in its move.

1) It checks all the horizontals (rows). If, anywhere, there are two bot tokens in the same row and the third spot is vacant, it fills that spot.

2) If no horizontal victories are available, it checks every vertical (column). If, anywhere, there are two bot tokens in the same column and the third spot is vacant, it fills that spot.

3) If no vertical victories are available, it checks both diagonals. If, anywhere, two bot tokens are in the same diagonal and the third spot is vacant, it fills that spot.

Checking for Losses

The next priority for the bot is to make sure it doesn't lose in the next turn.

Similar to checking for wins, the bot checks using the same patterns, but instead of checking for its own tokens, it checks for the user's tokens, and fills the vacant spot.

Optimum Strategy

If the game cannot end in the current or the next move, the bot can now set the tone for the game. Its strategy depends on a number of factors:

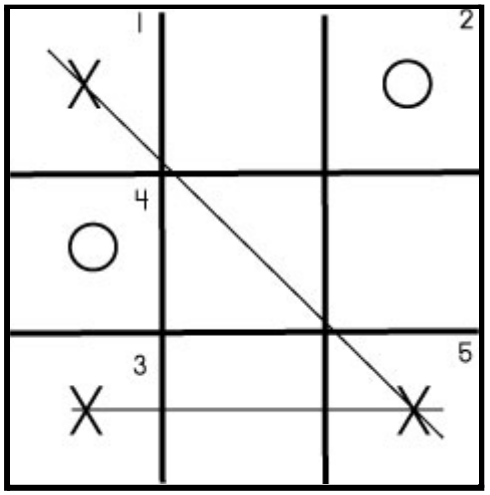
1) Current board setup
2) Number of turns
3) Bot token (X or O)
4) Final objective (Attack or Defense)
5) Availability of "Key Spots"

BOT STRATEGY AS ‘O’

As ‘O’, the primary purpose of the bot is to defend the game and play for a draw. Winning as ‘O’ is effectively impossible, especially against the average player. As a result, the strategies as ‘O’ are relatively simple.

Playing as ‘O’ is a game of avoidance and blockage, so the priority is to just deal with the current turn, instead of having to think about future turns.

As a defendant, the algorithm that checks for losses and blocks them takes care of the turn-by-turn work. The only way for the user to win, as a result, is to set up the board in a way that they can have two different winning opportunities in the same turn. So, even if one of them gets filled, the user has another victory on the board to use. We will refer to this as a “Double Play’.



In the given figure, X has set up an example of the aforementioned Double Play. In this situation, ‘O’ can either block the bottom row, or the left diagonal. However, blocking one leaves the other open.

So, it can be said that a successful Double Play is a guaranteed win.

Due to the fact that Double Plays require vacant space, and vacant spaces fill up with every turn, they must be set up in the first 4-5 turns.

So, for the first few turns, the objective of the bot playing as ‘O’ is to block all setups of Double Plays. As long as they are blocked, the game will have to end as a draw.

To play the most secure game, the bot always plays the center square if available, if no strategies are involved.

Key Spots

The two “Key Spots”, from the perspective of ‘X’, are the center square and any one corner square. Starting in any of these positions gives an easy setup to take control of the board and set up Double Plays. Defending strategies differ for each key spot.

X takes Center

If X takes the center, the only way to defend against a Double Play is to take a corner (choice of corner is irrelevant). The bot always selects the top-left corner.

If the user takes **any choice except the opposite corner** (bottom-right corner), the blocking (loss detection) algorithm comes into play, and the game inevitably results in a draw.

If the user selects the bottom right corner, the bot only has to select any of the remaining corners to end the game in a draw. Selecting any of the side-squares (squares adjacent to the center) results in a Double Play.

So, the bot selects the top-right corner, resulting in a draw, and a successful defense.

X takes Corner

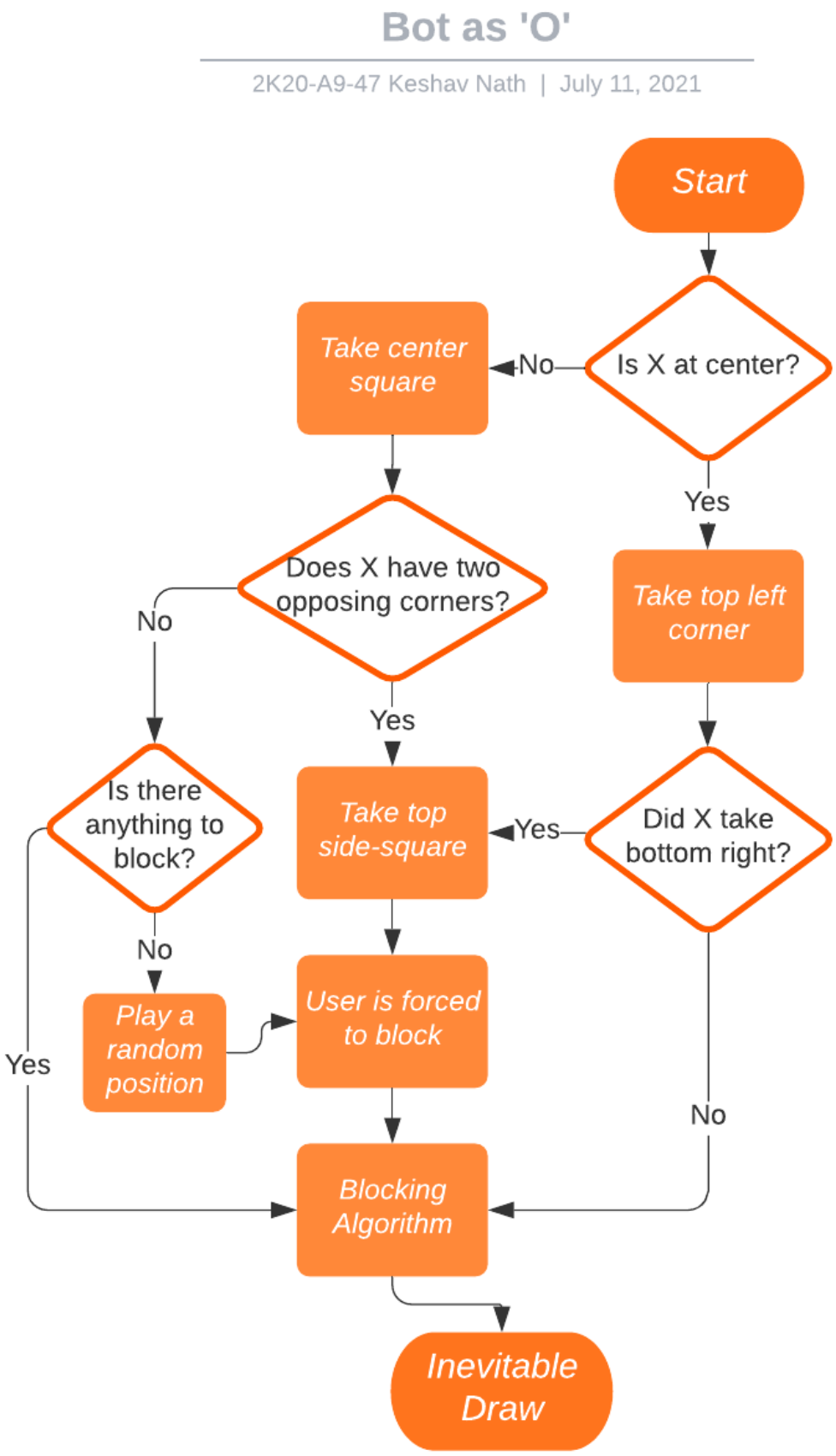
If X takes any corner, the best strategy is to immediately take the center square. This eliminates a number of possible Double Plays.

If the user takes **any choice except the opposite corner**, the blocking (loss detection) algorithm comes into effect, and the game eventually ends as a draw due to a series of blocks.

If the user selects the opposite corner, the bot has to select any side square to create a draw. Selecting a vacant corner will create a Double Play for X.

So, the bot always selects the top side square (square on top of the center).

Logical Flowchart



BOT STRATEGY AS ‘X’

As ‘X’, the primary objective is to play aggressively, create Double Plays, control the board, and let the user do the blocking.

The strategies of ‘O’ were thoroughly discussed in the previous section, and the strategies of ‘X’ are exactly the opposite. In this case, the user plays as a defendant and assumes the roles of ‘O’ discussed above.

Key Spots

As discussed previously, the Key Spots for X are the center and the corners. The game is played around these squares.

So, the **bot always starts in either the center or the top-left corner.** This gives the best possible setup for Double Plays.

Starting from Center

The Double Play from the center square generally takes the following forms:

X^5		X^3
	X^1	0^2
0^4		

0^2		
0^4	X^1	
X^5		X^3

Here, the superscript numbers (1, 2, 3, 4, 5) represent the turn in which that token was played. After X takes the center square, the game can go one of two ways.

1 - ‘O’ Takes Side Square

If ‘X’ takes the center and ‘O’ takes a side square, as shown in the first image, the best strategy for the bot is to take a corner adjacent to that side. After the user blocks, the bot then simply has to take the corner that forms a free triangle (a triangle with an empty spot at the center). This creates a Double Play.

2 - ‘O’ Takes a Corner

If ‘X’ takes the center and ‘O’ takes a corner, X can only take the opposite corner. Any other selection results in a series of blocks that end in a tie.

Upon taking the opposite corner, the user has only two options:

1) Take any remaining corner square
2) Take any side square

Option 1) leads to a series of blocks for a tie.
Option 2) sets up the opportunity for a Double Play, often just by Blocking.

Starting from Corner

The Double Play from the corner square generally takes the following forms:

X^1	0^4	X^3
0^2		
		X^5

X^1		0^4
	0^2	
X^5		X^3

Here, the superscript numbers (1, 2, 3, 4, 5) represent the turn in which that token was played. After X takes the corner square, the game can go one of two ways.

1 - ‘O’ Takes Side Square

If ‘X’ takes the corner and ‘O’ takes a side square, the best option for X is to take the adjacent corner that is not touching the ‘O’ (as shown in the figure). The user will be forced to block, and then ‘X’ takes the corner opposite from the first corner. This creates a Double Play.

2 - ‘O’ Takes Center

If ‘X’ takes a corner and ‘O’ takes the center, X can only take the opposite corner. Any other selection results in a series of blocks that end in a tie.

Upon taking the opposite corner, the user has only two options:

1) Take any remaining corner square
2) Take any side square

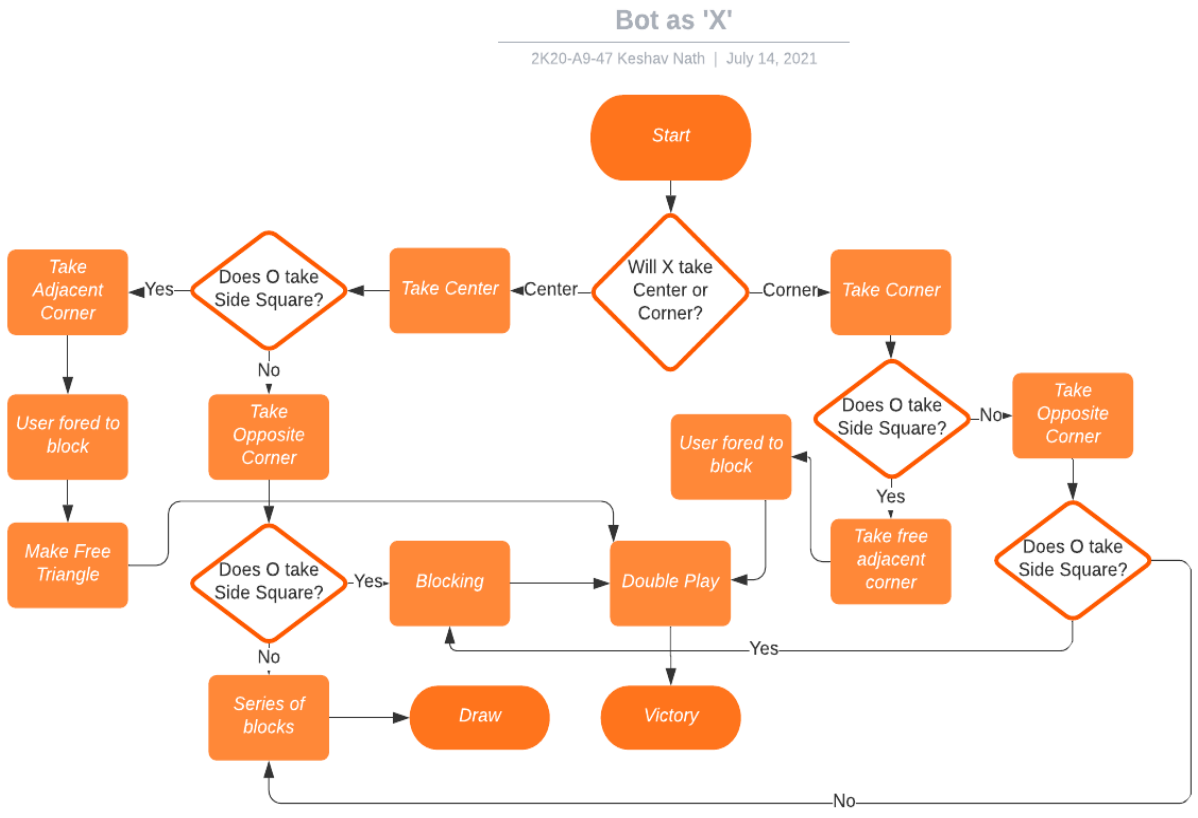
Option 1) leads to a series of blocks for a tie.
Option 2) sets up the opportunity for a Double Play, just by Blocking.

Strategy Decision

As seen above, the bot can start in one of two ways. Two different strategies were implemented to make the bot less predictable, and even harder to beat.

The bot takes a decision on which strategy to use (Center or Corner) completely randomly, as both are equally viable and hard to counter.

Logical Flowchart



GAMEPLAY SAMPLE

```
        0  1  2

     0  -  -  -

     1  -  -  -

     2  -  -  -

WELCOME TO SMARTBOT PROJECT by KESHAV NATH and VATSAL
MITTAL

We have created a BOT for Tic Tac Toe / Knots and Crosses
/ XO
It is nearly impossible to defeat!

Press 'X' key to be X or 'O' key to be O
Remember, X always starts first.

o
Press D to play against a dumb bot, or anything else to
play against the SmartBot!
1
Good Luck.

 COMPUTER STARTS


        0  1  2

     0  -  -  -

     1  -  -  -

     2  -  -  -
```

```
THE COMPUTER PLAYED AT 1,1

        0  1  2

     0  -  -  -

     1  -  X  -

     2  -  -  -


YOUR TURN
ENTER ROW :-2
ENTER COLLUMN :-2
YOU PLAYED AT 2,1

        0  1  2

     0  -  -  -

     1  -  X  -

     2  -  0  -

THE COMPUTER PLAYED AT 0,0

        0  1  2

     0  X  -  -

     1  -  X  -

     2  -  0  -


YOUR TURN
```

```
ENTER ROW :-2
ENTER COLLUMN :-2
YOU PLAYED AT 2,2

        0  1  2

     0  X  -  -

     1  -  X  -

     2  -  0  0

THE COMPUTER PLAYED AT 2,0

        0  1  2

     0  X  -  -

     1  -  X  -

     2  X  0  0


YOUR TURN
ENTER ROW :-1
ENTER COLLUMN :-1
YOU PLAYED AT 1,0

        0  1  2

     0  X  -  -

     1  0  X  -

     2  X  0  0

THE COMPUTER PLAYED AT 0,2
```

```
       0  1  2

    0  X  -  X

    1  0  X  -

    2  X  0  0



 ||  X  -  X  ||

 ||  0  X  -  ||

 ||  X  0  0  ||


COMPUTER WINS AS X


GOOD GAME
Press ESC to EXIT or ANY OTHER KEY to PLAY AGAIN.
```

Github Link - [keshavnath/tttbot: A TicTacToe Bot made in C (github.com)]

MOTIVATIONS

The creation of this project was taken up as a challenge, to create something “smart” without advanced techniques or Machine Learning. This bot simplu utilizes compounded if-else statement blocks.

Since Tic-Tac-Toe is a (relatively) simple game, with fewer variables, and most importantly has a set of verified “strategies” to play the game, it was selected as the target for our work.

Due to the symmetric nature of Tic-Tac-Toe (the fact that not all permutations of the board are actually unique), strategies are easy to implement with stabilization. For example, when playing as ‘X’ and using the corner strategy, we need not implement the strategy for all four corners. We can simply select one corner and work from there, as playing from any other corner is absolutely identical.

CONCLUSIONS

We have successfully created a “smart” Tic-Tac-Toe bot that proves to be impossible to defeat, and extremely difficult to defend against.

This task has been accomplished using only the C Programming Language and its standard libraries. This allows it to be compatible with a host of programs and operating systems without excessive preconfiguration.

We have also completely wrapped the bot in a playable Tic-Tac-Toe Game to perfectly demonstrate the detailed working of the bot.

FUTURE WORK

This project is elementary and has a lot of scope of improvement. Moving forward, the main areas of improvement would be:

1) Finding an implementing more winning strategies

   While the bot's defence is strong, it only uses two main strategies in the offence, both of which can be countered by fairly experienced players.

2) Increasing Shareability/Modularity

   In its current form, the project has been directly wrapped in a Command Line Tic-Tac-Toe Game, decreasing its reusability. We can instead wrap all of the functions in namespaces and create a header file out of the bot, so that it can be used in real-world projects.

3) Improving Efficiency

   Instead of using a host of if-else blocks, the board can be analyzed mathematically. This is helped by the fact that the board is represented as a 2D Array, and tokens are represented by different numerical values. This would result in a much faster bot.

4) Removing all Randomization

   The bot, when it has no strategies to implement, or when all the key spots are taken, choses a random spot (as it is assumed, by that time the result of the game has already been confirmed beforehand). This can be eliminated by introducing a deep dive into strategies and board analysis to ensure favourable outcomes (wins or draws).

REFERENCES

Beck, J. (2002). Tic-Tac-Toe. *Bolyai Soc. Math. Stud*, *10*, 93-137

Tic-Tac-Toe - Wikipedia - https://en.wikipedia.org/wiki/Tic-tac-toe

Hochmuth, G. (2003). On the genetic evolution of a perfect tic-tac-toe strategy. *Genetic Algorithms and Genetic Programming at Stanford*, 75-82.

Rapoport, A. (2013). *Two-person game theory*. Courier Corporation.

Myerson, R. B. (2013). *Game theory*. Harvard university press.

History of Tic Tac Toe - UC Berkeley GamesCrafters Research Group. (n.d.). GamesCrafters©. Retrieved July 14, 2021- http://gamescrafters.berkeley.edu/games.php?game=tictactoe

Programming Fundamentals
CO-102

Sem II - MTE Project

on

SMART BOT FOR TIC-TAC-TOE

Submitted by:
KESHAV NATH - 2K20/A9/47
VATSAL MITTAL - 2K20/A9/82

Under the supervision of
Ms. Priya Singh
Department of Computer Science
Delhi Technological University

THANK YOU