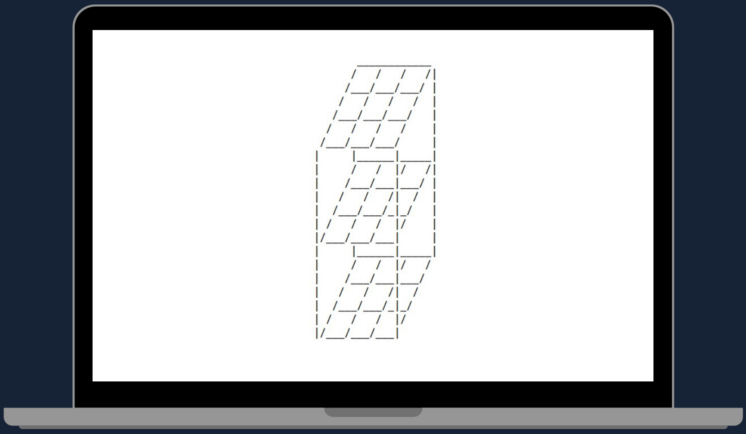
DISCRETE MATH || CS-1104

3D TIC TAC TOE

Implimentation of 3D tic tac toe with 1v1 multiplayer and 1vComp modes



Aditi Jain
UG22

Aditya Sarkar
UG22

Arya Nagar
UG22

Tanvi Roy
UG22

APRIL 23, 2020
SEMESTER PROJECT

3D Tic tac Toe

Aditi Jain Aditya Sarkar Arya Nagar Tanvi Roy

April 23, 2020

Website and Repositories (links):

Code Repository Game Webapp Website Repository

1 Introduction

Tic-tac-toe (noughts and crosses or Xs and Os) is a paper-and-pencil game for two players who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks (a 3-in-a-row) in a horizontal, vertical, or diagonal row is the winner.

Three-dimensional (3D) Tic-tac-toe is a more complex version of the classic two-dimensional Tic-tac-toe game. It can best be imagined as having 3 two-dimensional Tic-tac-toe boards stacked one on top of another. The goal of getting a 3-in-a-row remains intact - the first player to mark 3 squares in a row as 'X' or 'O' wins. However, there is an additional level of complexity by having 3 two-dimensional Tic-tac-toe boards instead of one for the total number of positions increases from 9 to 9^3 = 27.

2 Problem Description

The problem we undertook was "Implement a three-dimensional Tic-Tac-Toe. Similar to the two-dimensional game, three X's or O's in a row decides the winner and thus ends the game." We also developed a 2 player mode and 2 bots for a single player mode.

3 Solution Design

We thus began our project by understanding the complexity in 3D Tic-tac-toe. First, we calculated all winning positions or 3-in-a-rows possible. Normally, in a single 2D Tic-tac-toe board, there are 8 possible 3-in-a-rows (3 horizontal, 3 vertical and 2 diagonal). In a 3D Tic-tac-toe game, we calculated the total number of 3-in-a-rows to be 49. We arrived at this number by doing the following:

8 from the top most layer + 8 from the middle layer + 8 from the bottom most layer + 9 vertical straight columns joining each of the 3 layers + 8 outer diagonals + 8 inner diagonals.

3D Tic-tac-toe is a n^k game, where
n = number of dimensions
k = length of the edge of the cube
Thus, the number of winning lines for a cube of edge k in n-dimensional space =

$$\frac{(k+2)^n - k^n}{2}$$

For a 3D Tic-tac-toe cube, n = 3 and k = 3. Therefore, we get:
$\frac{(3+2)^3 - 3^3}{2}$ = 49, which verifies our aforementioned claim.

We came up with 2 functions. One would take in an integer value (from 1 to 27) as input which would correspond to a particular position in the game. This function will be called every turn.
The 2 players keep making moves, until either one of them wins, or all positions are taken and the game ends in a draw.

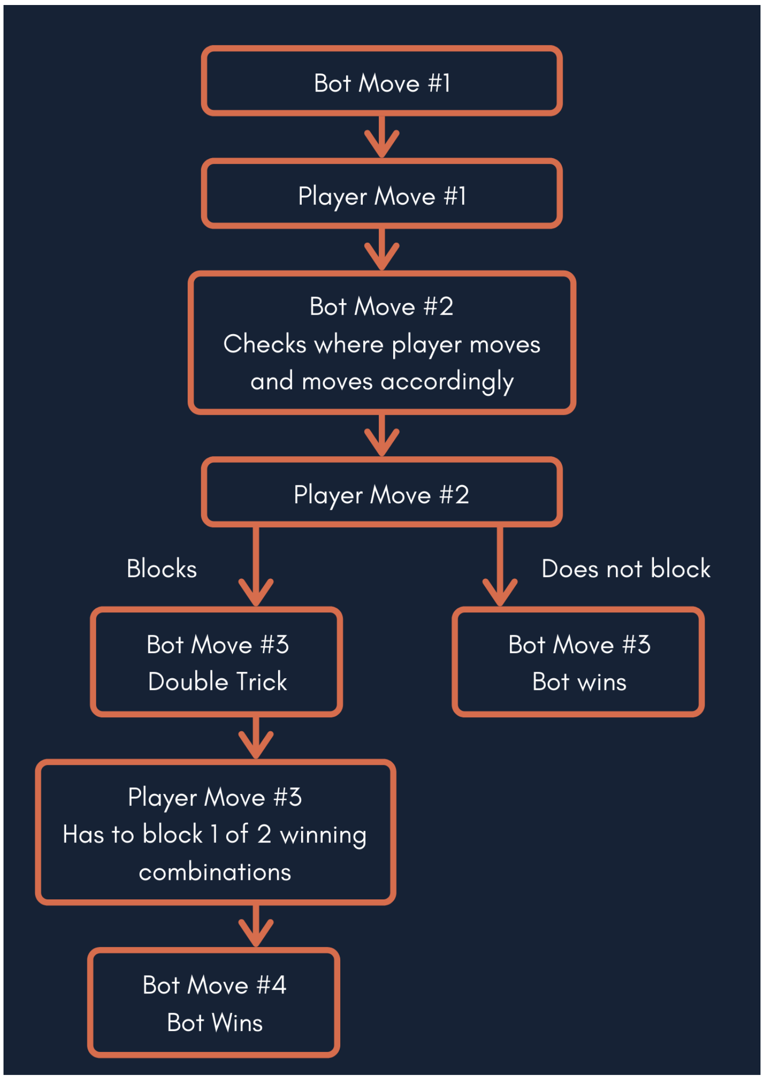


Figure 1: Flowchart of Bot, when bot plays first

3.1 Bot: If bot starts the game

The flowchart explains how the algorithm helps the bot work and win by the end of a maximum of seven moves. After the bot makes the first move for the computer at position 14, the player moves. After accessing where the player has moved, it adopts a particular game plan to make the next move. After the second move of the bot, the second move of the player decided whether the game ends here or not. If they don't block the computer, the game ends. If they end up anticipating the computer's win, the bot uses a double trick in its third move to trick the player into losing by the seventh move. Regardless of where the player places their third move, the bot will win by its fourth move i.e. the overall seventh move.

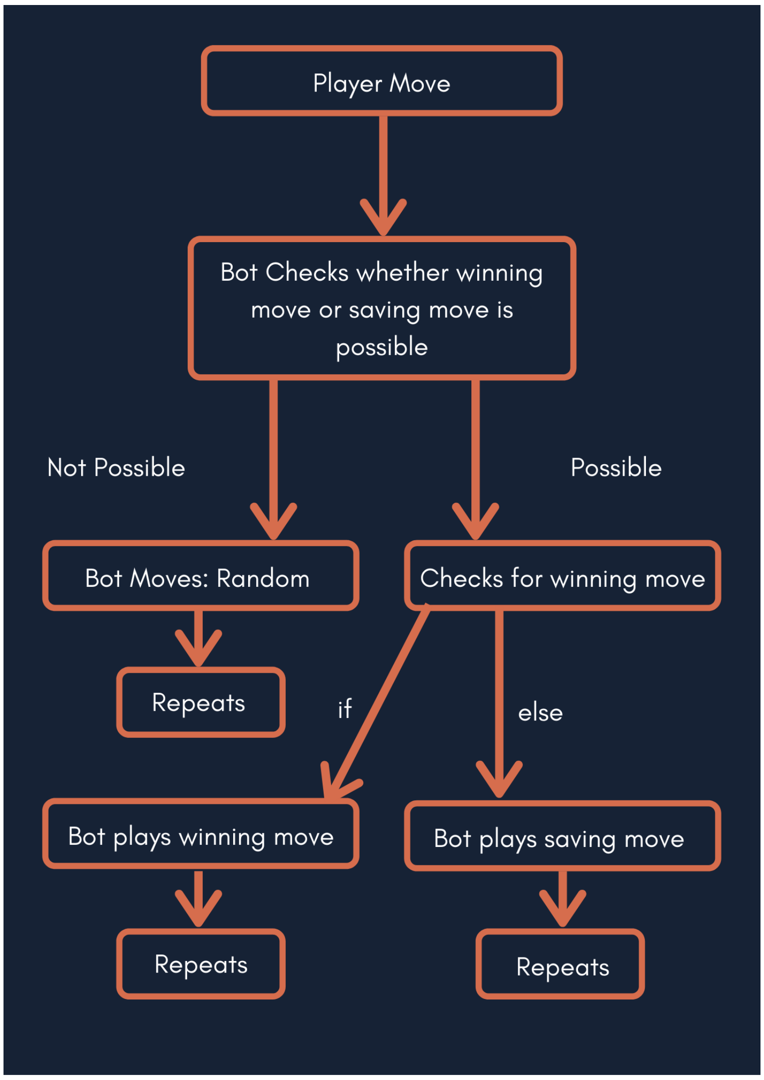


Figure 2: Flowchart of Bot, when Player plays first

3.2 Bot: If player starts the game

As we can see from the flowchart, the bot always responds to the move made by the player. Firstly, it will check if there is a combination where the bot can win. If not, it will check if there is a combination for which the bot has to make a move which ensures that it doesn't lose. The bot always checks the winning combination before the saving combination. If neither of the combinations are available, the bot will randomly give any move on any of the remaining unoccupied spaces.

4 Solution Implementation

We chose to build our version of the game by coding it in Python - a versatile programming language that each of the group members were familiar with.

4.1 Game Architecture

The game architecture can be easily understood by viewing the 1v1 game for the 1vComp games are its adaptation where one of the players has been replaced by the bot.

Each of the value of the $27(3 \times 3 \times 3)$ spaces of the tic tac toe board are stored in an Array of length 28 with a leading empty space so that the input values correspond with the index of the Array. This array is displayed in an ASCII visualisation by a function called `Display(Array)`.

In the 1v1 mode, player 1 plays first using the `TurnX` function which calls the `InputX` function by defualt. The `InputX` function recursively calls itself until a valid input is provided. On receiving a valid input, the corresponding index in the Array is turned to to X.

The game then proceeds to check if player 1 has won by calling the `checkwinX` function which checks for all 3-in-a-row winning combinations. If player 1 has won, the game is now. If not, the function calls the `checktieX` function which checks if all spaces (1-27) all filled in the Array to check for the condition for a draw. If all spaces are occupied, the game ends on a draw. Otherwise, the game proceeds to player 2's turn which takes place in a fashion similar to player 1's turn. This cycle continues till either player 1 or player 2 have won, or the game is a draw.

4.2 Bots

We designed and implemented two different bots for the game depending on if the player goes first or second, which is decided by a coin toss.

4.2.1 Bot: if player starts the game

On the basis of the coin toss, this particular bot is invoked. Since the human player makes the first move, the primary aim of the bot is to analyse the situation after every move that the player makes. Accordingly, the bot decides its move to get the best possible result.

As we have derived before, the total number of winning combinations are 49. Overall, the bot makes its moves based on two functions: `WinningMove()` and `Savemove()`. After the player makes a move, the bot checks if there is a move that will assure its victory. If not, it will check if there is a move that it can make to save its game by making sure that the player does not win. If neither of the situations occur, the bot will play a random move.

Lines 105-449: The `Winningmove()` function here checks all the combinations where the bot can assure a victory. If it finds a combination with an empty space where the bot can make a move and win, it returns that position value to K, which stores the winning move. If there is no winning combination available, the function returns 0.

Lines 454-798: If the value of K returned by the `Winningmove()` function is 0, the `Savemove()` function is called. Similar to the `Winningmove()`, the `Savemove()` function checks if the bot has to play any move to ensure that it does not loose. If such a case exists, the function returns the position as K.

In both cases, if the there is no such combination available, the bot simply makes a random move on any of the remaining spaces.

4.2.2 Bot: if bot starts the game

This bot is for when the computer plays first, which is decided by the coin toss. This particular bot has been designed so that there is a guarantee of the computer winning in seven or less moves, regardless of where the player makes their move.

Lines 1128-1136: The first move of the computer and the game is set to position 14 which is the centre position of the centre 2D tic-tac-toe. In 2D tic-tac-toe, playing at the centre position gives a huge advantage which guarantees a win/tie. In a 3D tic-tac-toe, this probability is increased multi-fold due to the various combinations of winning. The bot has been designed in a way that eliminates the possibility of a loss/tie for the computer.

Lines 6-7: The entire game is based on the first move of the player which is why we have employed a global variable k which stores the position of the first move of the player. This global variable is only accessed to employ a separate game plan for whatever first move has been made by the player. Simultaneously, we have employed another global variable x which decides which move number it is. This is important for this decides the moves of the bot subsequent to the first move of the player. It has been initialised at 1 i.e. the first move of the computer. All of the odd numbers denote the move of the computer while the even number denote the move of the player.

Lines 1146-1152: After the computer has made its first move at 14, the player makes their move. In this, the places that the player can make their move have been segregated under three labels which can be seen in the `Xcheckposition()` function. The three categories are: `Xsidemove()` (positions: 11, 13, 15, 17, 5, and 23), `Xbodydiagonal()` (positions: 1, 3, 7, 9, 19, 21, 25, and 27), and the rest in `Xweirddiagnoal()`. To explain the algorithm, we have interpreted the 3D tic-tac-toe as three 2D tic-tac-toes stacked on top of one another.

Lines 1190-1229: If player's first move has been made at a side move position: If the move has been made on the first or third 2D tic-tac-toe layer of the 3D tic-tac-toe (positions: 5 or 23), the bot places its second move on the first 2D tic-tac-toe layer of the 3D tic-tac-toe. If the move has been made on the second 2D tic-tac-toe layer of the 3D tic-tac-toe (positions: 11, 13, 15, or 17), the computer makes its second move at a position on the second 2D tic-tac-toe of the 3D tic-tac-toe itself.

Lines 1232-1284: If player's first move has been made at a body diagonal position: If the move has been made on a body diagonal (positions: 1, 3, 7, 9, 19, 21, 25, or 27), the bot places its second move at a position adjacent to the position at which the player has moved. This is done on the same plane i.e. the first 2D tic-tac-toe layer of the 3D tic-tac-toe or the third 2D tic-tac-toe layer of the 3D tic-tac-toe according to the player's move.

Lines 1286-1338: If player's first move has been made at a weird diagonal position: If the move has been made on the first or second 2D tic-tac-toe layer of the 3D tic-tac-toe (positions: 2, 4, 6, 8, 10, 12, 16, or 18), the bot places its second move on the first 2D tic-tac-toe layer of the 3D tic-tac-toe at a position which is diagonally opposite from the position at which the player has played. If the move has been made on the third 2D tic-tac-toe layer of the 3D tic-tac-toe (positions: 20, 22, 24, or 26), the bot places its second move on the third 2D tic-tac-toe layer of the 3D tic-tac-toe at a position which is diagonally opposite from the position at which the player has played.

At this point, the player can either block the computer's winning move or play it somewhere else in case they are unable to anticipate the computer's winning move. If the person does not block the computer, the computer wins during its third move i.e. the overall fifth move. If they anticipate the computer's winning move, the bot employs the double trick method to win and thus conclude the game in the seventh move.

The entire algorithm of the bot has been based on the first move of the player. After the player's first move, the algorithm has defined a specific way which leads to the bot winning regardless of where the subsequent moves are made by the player. Using this method, the algorithm of the bot has been designed in a way that ensures that the computer wins every time in seven or less moves.

4.3 Integration

Integration was a material task for it involved working with different bot files and ensuring that they do not clash with one another.

The first version involved a python based if else statement that could determine the mode (Single player/Double player). The resultant mode corresponded to a number which triggered only those corresponding parts of the game architecture.

```
def menu():
    print("<<<<<  3DTTT >>>>\n\n\n")
    print("... Main Menu ...\n")
    print("1. One Player's Game\n")
    print("2. Two Players' Game\n\n")

def Main():
    Mode=0
    menu()
    while(Mode!=1 and Mode!=2):
        Mode=int(float(input("Enter your Choice: ")))
    if (Mode==1):
        toss(Mode)

    else:
        DisplayMaze()
        TurnX(Mode)
    print("\n\n\n\n")
```

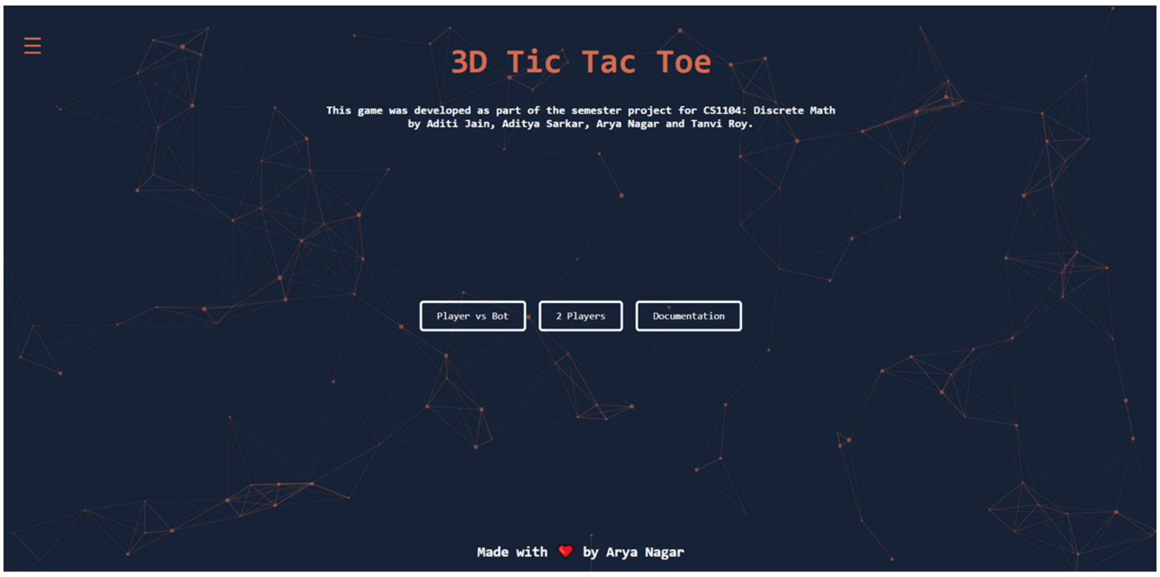
This was later replaced by the website which takes 2 different modes and shows them on 2 different landing pages, thereby simplifying the code.

Toss: One of the suggestions given by Professor Paul was to implement a coin toss to decide who plays first (bot or player) for the single player mode. Thus, the following code was implemented to integrate toss into the game, the output of which starts either of the game modes.

```
def toss():
    choose=0
    print("  ______               ")
    print(" /_  __/___   __________")
    print("  / / / __ \ / ___/ ___/")
    print(" / / / /_/ |(__  |__  ) ")
    print("/_/  \____/ ____/____/  ")
    print("\n")
    print("Choose H/T\n\n")
    print("Winner plays first\n")
    print("Enter 1 to select HEADS\n")
    print("Enter 2 to select TAILS\n")
    while(choose!=1 and choose!=2):
        choose=int(float(input("Enter your Choice: ")))
    check = random.randint (1,2)
    if check==choose:
        print ("PLAYER GOES FIRST")
        Display()
        TurnX()
    else:
        print ("COMPUTER GOES FIRST")
        print ("NICE")
        Display()
        Xturnbot()
```

4.4 Website(Final Product)

Once we finished the final python game files, we decided to host the game on a website for easy access and for a fun visual representation. We coded a webpage with the two game modes as two landing pages and then we embedded the python game files into the webpages using Trinket.io



3D Tic Tac Toe

This game was developed as part of the semester project for CS1104: Discrete Math by Aditi Jain, Aditya Sarkar, Arya Nagar and Tanvi Roy.

Made with ❤ by Arya Nagar

Figure 3: Screenshot of Game webpage

5 Timeline

Stage 1: Arya made the initial game architecture. Tanvi found out the winning combinations and designed the CheckWin function, which Arya implemented in the code.

Stage 2: Aditi designed the algorithm for the bot in the player vs computer mode when the computer starts the game, which was then coded and implemented. Aditya designed the bot for the player vs computer mode when the player starts the game. These were made compatible with the existing game architecture by Aditi and Arya.

Stage 3: Tanvi and Aditi worked on the visualisation of the game using Pygame (Aditi on the 3D visualisation of the cube and Tanvi on the rotation of the cube) but it could not be successfully implemented in this project.

Stage 4: Arya modified the code and coded the website for the game with ASCII visualisation.

6 Conclusion

While there are many different Tic-tac-toe projects that exist on the internet, our group made a genuine effort to come up with a working 3D tic-tac-toe game from scratch, and have 2 fully functioning modes the user can choose to play. A genuine effort was made by each group member to experiment and innovate, and we learned several things while collaborating to come up with this project.In trying times like these the project was a quiet retreat to keep our minds from boredom and a means to learn something new

7 Memberwise Contribution

1. All
   - Documentation (on shared LaTeX project)
2. Aditi Jain
   - Designed `bot` for player v/s computer when the computer starts the game
   - Designed and implemented own algorithm for above `bot`
   - Coverted `bot` code to be compatible with game architecture
   - Ineffectual: Worked on Pygame 3D visualisation (Not included in final project)
3. Aditya Sarkar
   - Designed bot for player v/s computer when the Player starts the game
   - Coverted bot code to be compatible with game architecture
   - Made initial "Main Menu" which was later removed during front end adaptation
4. Arya Nagar
   - Made the game architecture which is the backbone on which the 2 bots are later added
   - Made the 1v1 Game
   - Maintained versions
   - Standardized all code so that it is compatible with the rest of the project
   - Wrote and Implimented Final code
   - Designed visualisation and coded the website for the game
5. Tanvi Roy
   - Wrote the `Checkwin` function in the game architecture
   - Ineffectual: Worked on Rotational Matrices Pygame implementation (Not included in final project)

8 Citations

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

http://pi.math.cornell.edu/ mec/2003-2004/graphtheory/tictactoe/tttanswer1.html

https://github.com/VincentGarreau/particles.js (For website)