**Problem Definition and Analysis**

For my final project, I have chosen to program the game “Tic Tac Toe” using Python. This choice was motivated by my general interest in games and interest in understanding their structure and underlying code; Python was chosen as the programming language studied during the course “Computer Science”. This classical game, with its simple rules and structure, is a good choice for delving into programming and gaining practical experience with applying the skills learned in the course.

My project work included the development of a game for 2 human players with an input system with a mechanism to prevent invalid players’ actions, and a system that determined the outcome of the game: win X, win O, or draw. The program was written with functions (def), loops if/elif/else, while, and for; numbers were closely related to the game; therefore, the commands for integers were used.

**Documented design**

*Rules for the game:*

Tic Tac Toe is a turn-based game for two players. Players make moves, placing their symbols (X or O) on a 3\*3 board. The first player who lines up three symbols in a column, line, or diagonal wins. If all nine cells are filled without a winner, the game ends in a draw.

*Game structure:*

Objectives:

1. Create a working two-player Tic Tac Toe game.
2. Handle invalid inputs and prevent input into occupied cells.
3. Display a clear visual board.
4. Detect wins and draws correctly.

The game was implemented with four main functions:

* Drawing the game board
* Accepting players’ inputs, checking their correctness
* Determination of the output: the winner, or if it was a draw
* The main work of the mechanisms, showing the board, determining turns, and printing messages about the output of the game.

Therefore, 4 main modules were created using the functions def:

draw\_board(): Displayed the state of the board.

take\_input(): Handled the user’s input, checked correctness, and updated the board.

check\_win\_draw(): Checked for all possible winning combinations and determined the outcome.

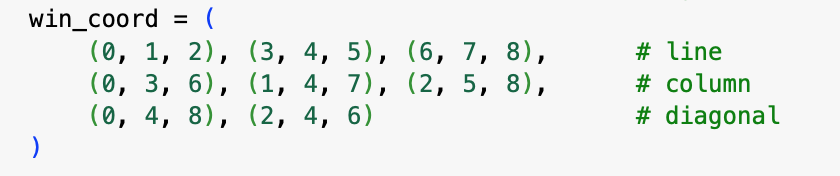
turn(): Coordinated the game, switched turns between players, and determined the game result.

*Initial data items*

board: A list of integers initially representing empty cells with numbers from 1 to 9 (positions to put ‘X’ and ‘O’), which later contained 'X' or 'O'



win\_coord: A tuple of tuples containing winning index combinations



counter: An integer that tracked the number of turns taken (used to check for a draw).

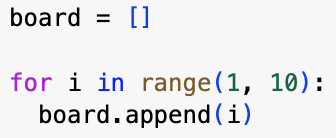
player\_answer: Initially, a string that stored the user's input for a turn. It would be converted into an integer if it met all conditions for valid input.

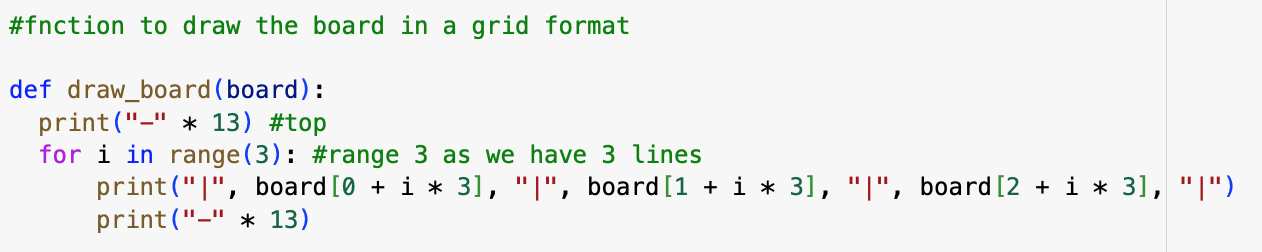
*Algorithms*

* Board

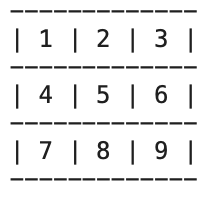
First, I programmed a board where players could place their tokens. A list board was created to represent this board.

Then, I filled this list with numbers from 1 to 9 (representing the positions on the cells), using a for loop.



After creating boards, a function draw\_board() was written, using a for loop.

This function printed the board in 3 rows. Each row showed 3 numbers with vertical bars separating them, and horizontal lines to make the grid clear.



* Input

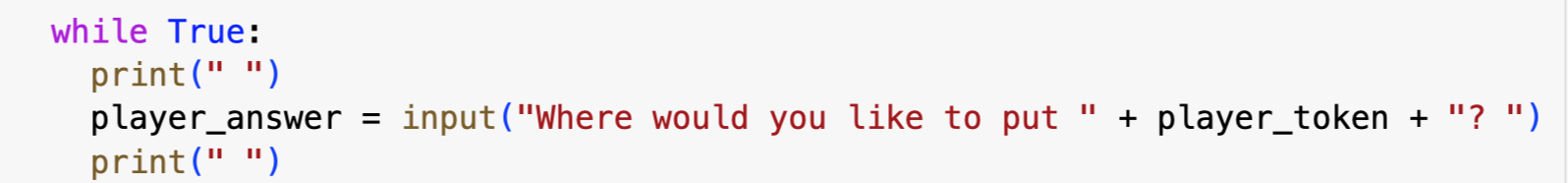
Once the game board was created, the program needed a way for players to make their moves.

For that, a function was created that asked players for a position on the board where they wanted to place their token. The foolproof was also included in this function.



The function took one argument called player\_token, which was either "X" or "O".

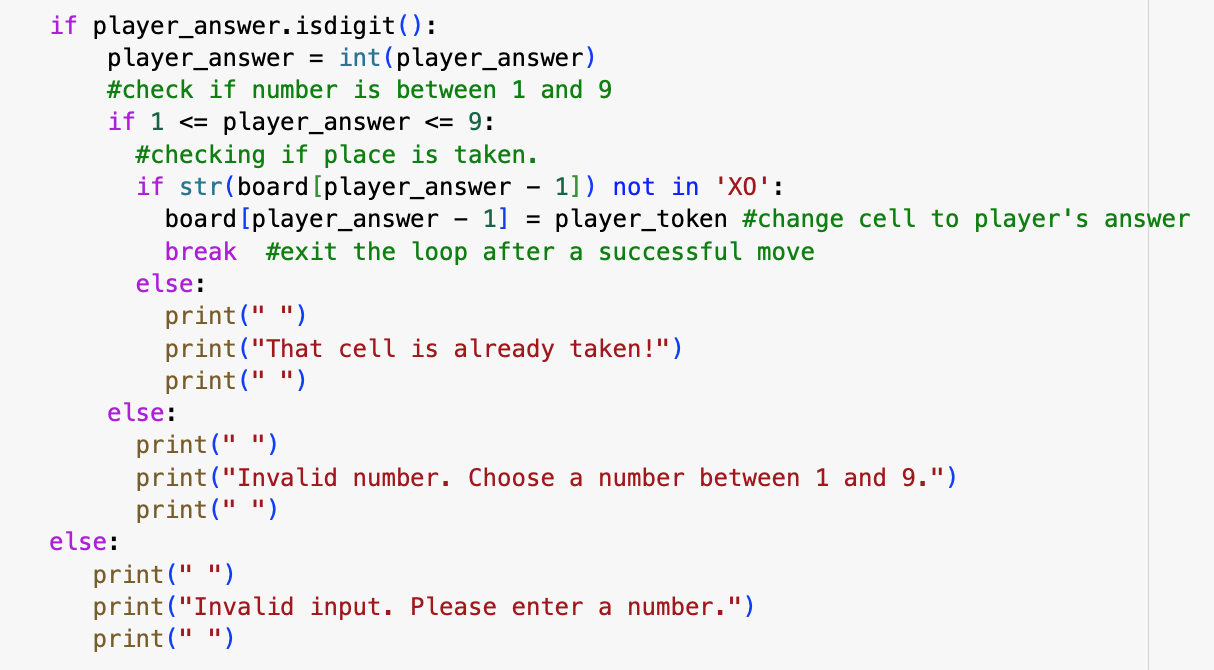
The program was supposed to keep asking the player for input until they entered valid input. For that, a while True loop was used. This type of loop runs forever unless a break statement is used to stop it.



Then, using if loops, it was checked if it was used, not another sign; and if so, it was converted into an integer for further checking for being between 1 and 9, and if the cell is taken.

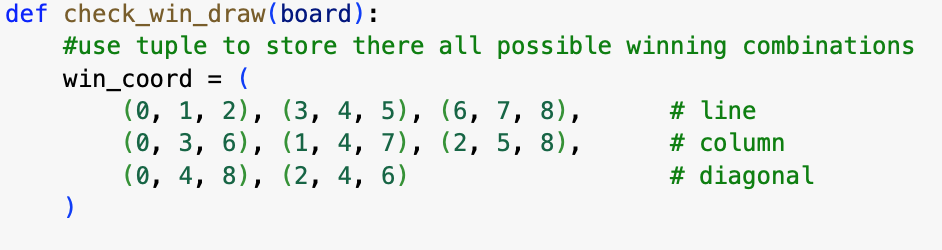
If the number was in the correct range, then it was checked whether the chosen cell was already taken. It was done by looking at the value in the board list at that position. If it was already "X" or "O", that meant someone had played there before.

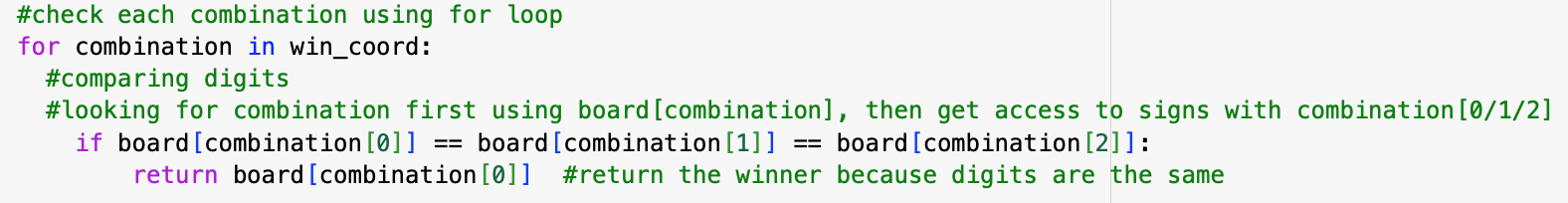
board[player\_answer-1] was used because the shown board started from 1 and not from 0. board[player\_answer] would give the next number, not the one the player wanted.



* Win Checker

To check if anyone had won the game, a function called check\_win\_draw(board) was created. Inside this function, a list of all possible winning combinations was made. Each combination was a set of three positions on the board that, if filled by the same player, meant a win.



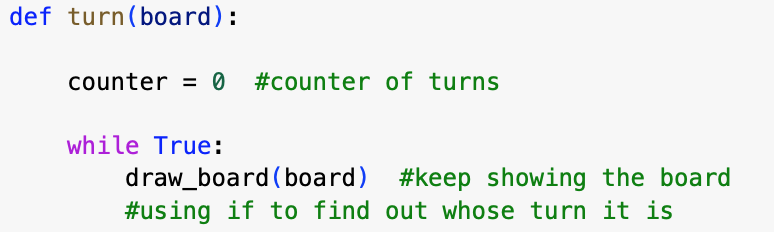
Then, a for loop was used to go through each of these winning combinations. For every set of three positions, it was checked if the same symbols were in all of them (a loop if was used for it). If so, that symbol was returned to show who won.

* Main mechanism

Function turn() was the ‘heart’ of the game.

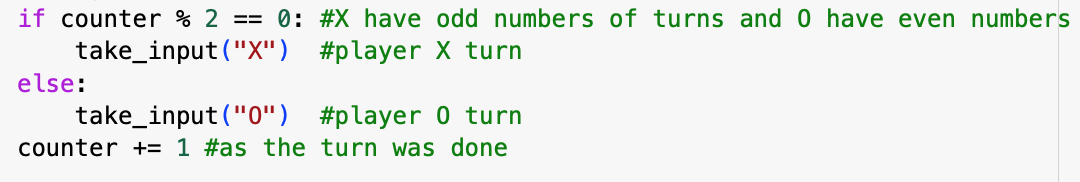
First, I created a variable counter, initially equal to 0. This variable was used to count how many moves had been made in total by that point in time.

Then, a while True loop was used to keep the program running as long as no one won or there was a draw. This type of loop runs forever unless a break statement is used to stop it.



In this loop, the draw\_board() function was called to represent the state of the board to let players see where they could move.

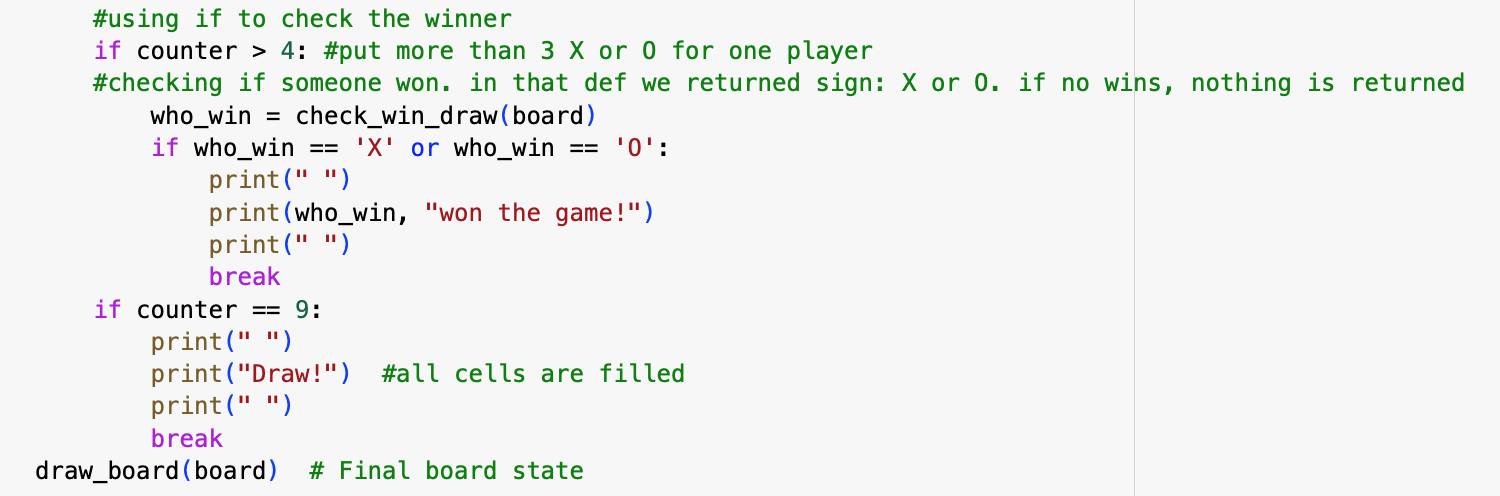
After that, the counter was checked to see whose turn it was, using an if loop. If the number was even, it was player X’s turn; otherwise, it was player O’s move. After, the take\_input() was called to let the current player make a move, and the counter was increased by 1 due to the move.



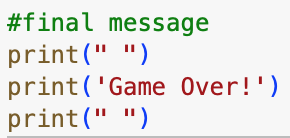
If at least 5 moves had been made (because that's the minimum needed for a player to win), the check\_win() was called to check if there was a winner. If so, the message saying who won was printed, and break was used to exit the loop.

If the counter reached 9 and no one had won, the message ‘draw’ was printed, because all the cells were filled.

After that, the final state of the board was printed.



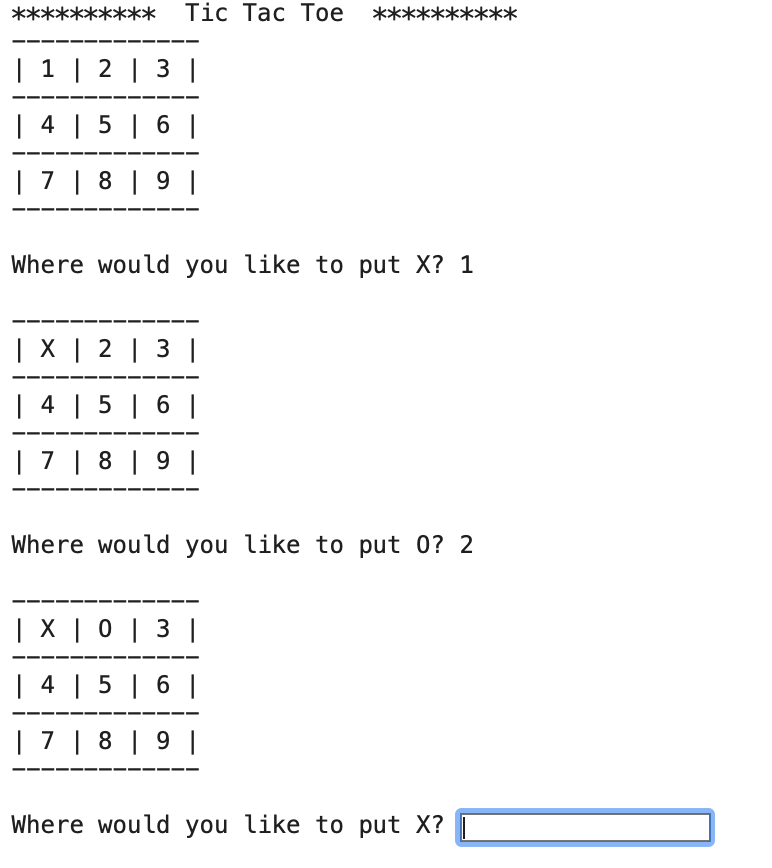
At the end of the game, the final message was printed.



**Tests**

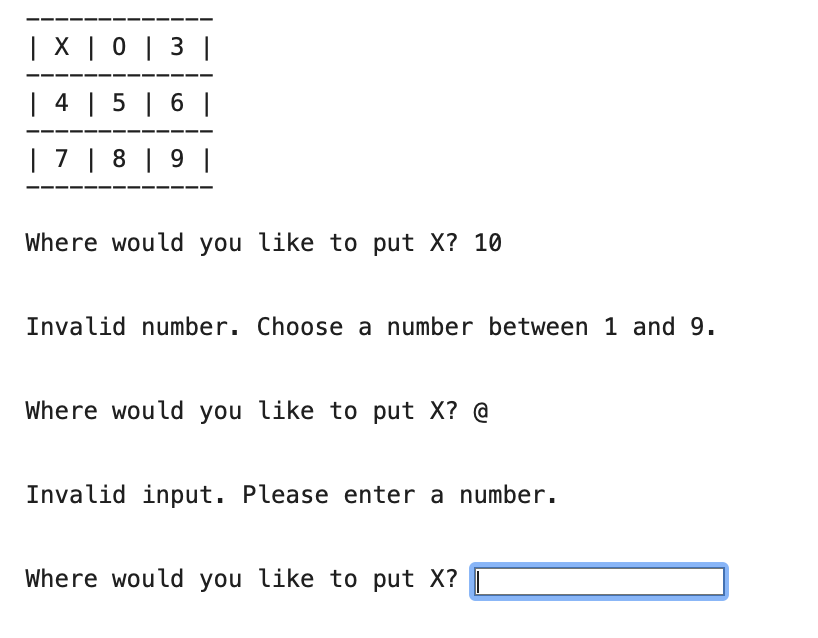
1. Valid input tests

As it was seen, the program allowed input of valid numbers and replaced them with X and O.



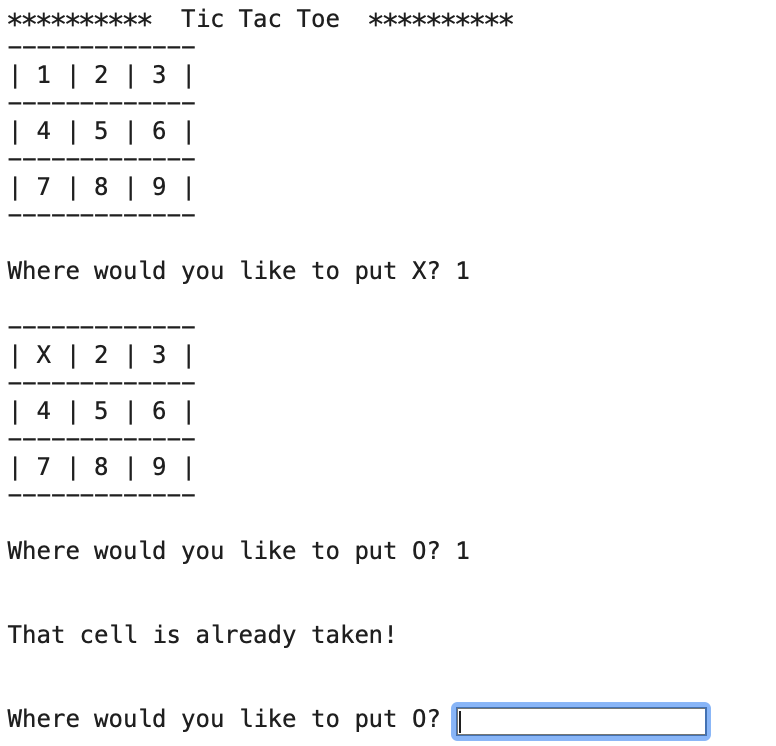
1. Invalid input tests

The next test showed that the program allows input of invalid signs and numbers, but printed the message and asked for new valid input.



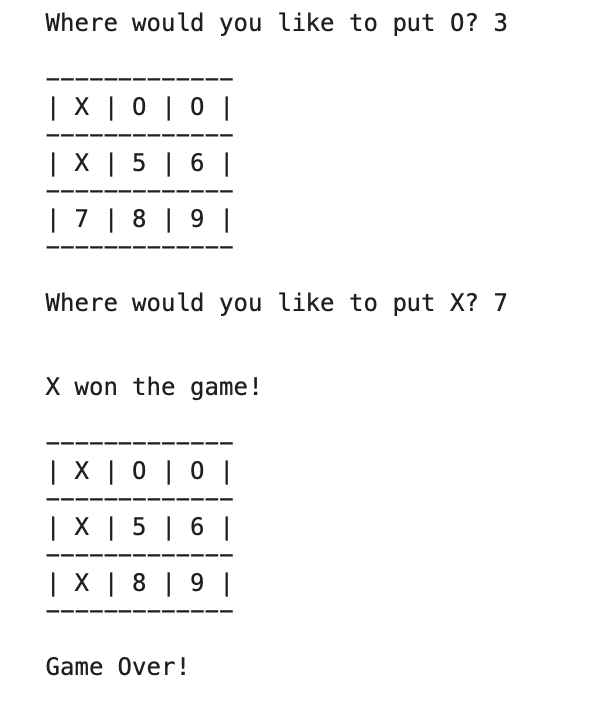
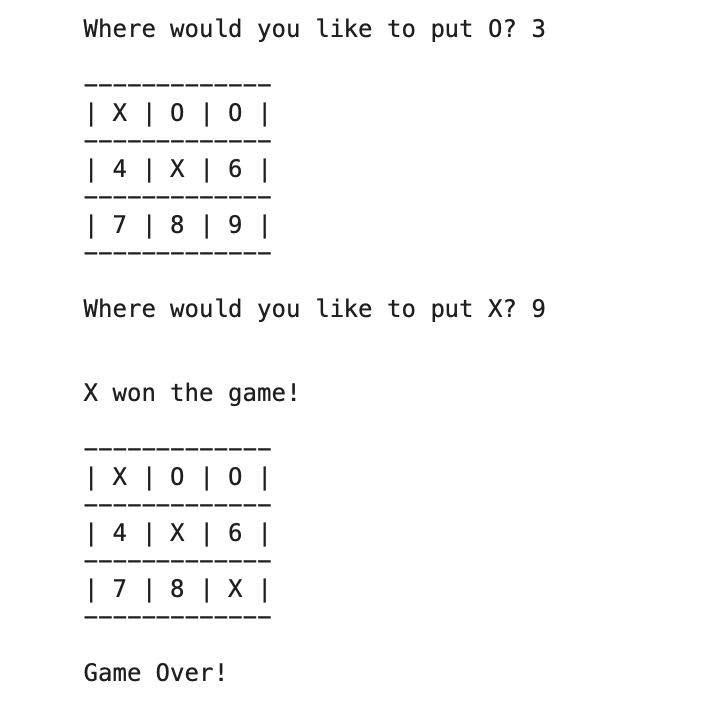
1. Occupied cell

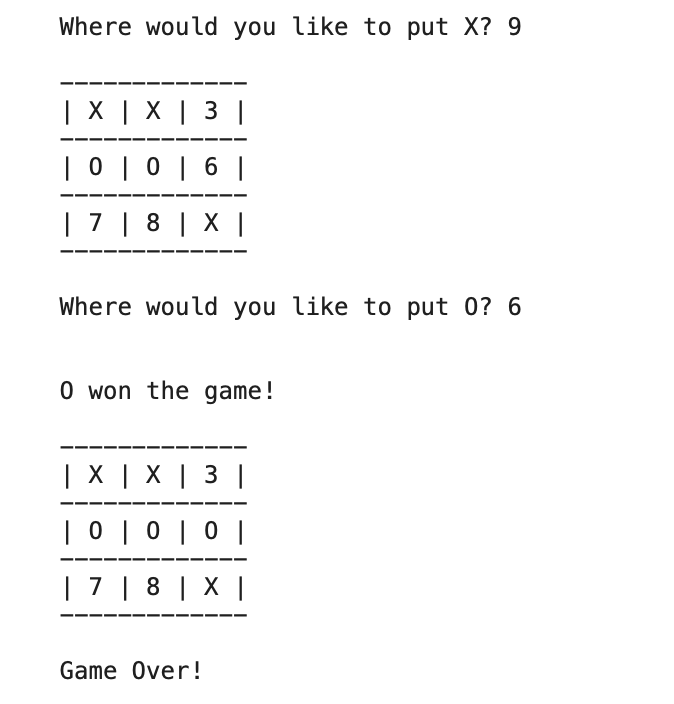
The system correctly prevented placing a symbol on an occupied cell.



1. Win combination

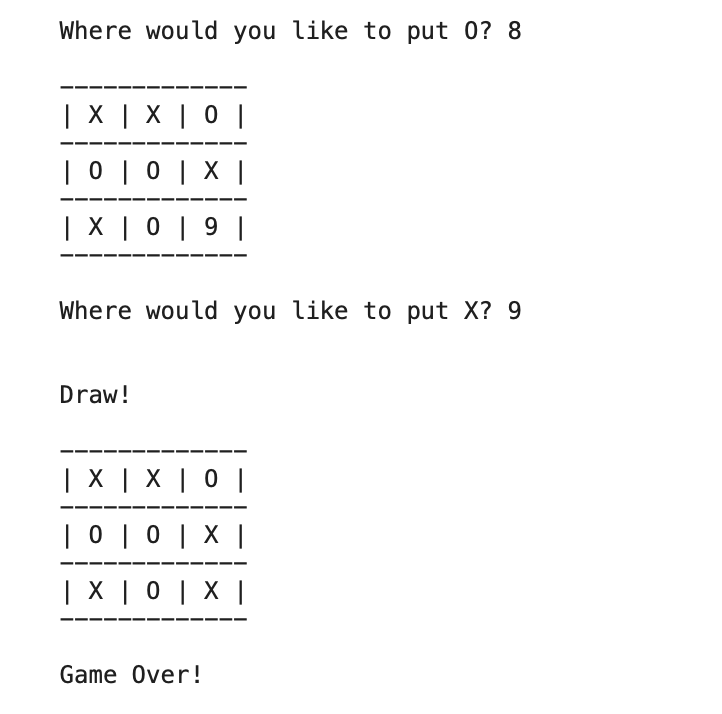
The last turns of the game show that the program accurately detected winning all combinations across rows, columns, and diagonals.





1. Draw test

When all cells were filled and no player had won, the game was declared a draw.



**Evaluation**

All goals were achieved effectively. The game was smooth, the board was clear, the foolproof worked accurately, and the winning and draw logic was accurate. One possible improvement would be adding the option of playing with the computer. Moreover, future improvements could include a graphical interface or score tracking.