**Project Name:**

ConnectFour Game

**Team Members:**

Zainab 24k-3080

Syeda Areesha Ahmad 24k-3043

# **Table of contents**

* **Introduction**

Background

* **Game Logic**

Grid Size

Rules and Winning Conditions

Move Processing

End of Game

* **Development**

Inputting Player Details

Grid Options Function

Dynamic Memory Allocation

Turn Check Function

Win Check Function

* **Implementation Details**

Horizontal Search

Vertical Search

Left Diagonal Search

Right Diagonal Search

* **File Handling**

PrintScore Function

UpdateScore Function

* **Testing**

Testing Different Grid Sizes (4x4, 5x5, 6x6)

Edge Cases

* **Challenges Faced**

Dynamic Memory Allocation (DMA)

Turn Check Issues

Search String Placement

* **Conclusion**

# Introduction

**Background:**

Connect 4 is a classic two-player strategy game where players take turns dropping colored tokens into a vertical grid. The objective is to form a horizontal, vertical, or diagonal line of four tokens before the opponent does. The game's simplicity and strategic depth make it an ideal project for applying programming concepts. This project implements a simple **console-based, two-player game** of Connect 4 where players interact through a text-based interface for their turns.

Game logic

**Grid Size:**

* The grid size is **dynamically allocated** based on user input.
* The grid's dimensions (rows and columns) will be specified by the player when starting the game, allowing the game to adapt to any size grid.

**Rules and Winning Conditions:**

* The rules remain similar to the Connect Four game: players take turns to drop their discs in one of the columns.
* A player wins if they achieve a horizontal, vertical, or diagonal line of **four consecutive discs** (this part does not change based on grid size).
* The grid will now have **dynamic dimensions**, but the **winning condition (four in a row)** stays the same.

**How Moves are Processed:**

1. **Grid Initialization:**
   1. The program will allocate memory dynamically for the grid based on user input for the number of rows and columns. (4x4, 5x5, 6x6 grids will be available).
2. **Turn Logic:**
   1. Players take turns to input a column number (from 1 to the number of columns).
   2. After the player inputs the column, the program checks for the first available (empty) spot in that column, starting from the bottom row and moving upwards.
3. **Win Check:**
   1. After each move, the program will check for the win condition (4 consecutive discs in a row, column, or diagonal) in the dynamically allocated grid.
   2. The win checking process remains the same: horizontal, vertical, and diagonal checks, but the grid size is flexible now.

**End of Game:**

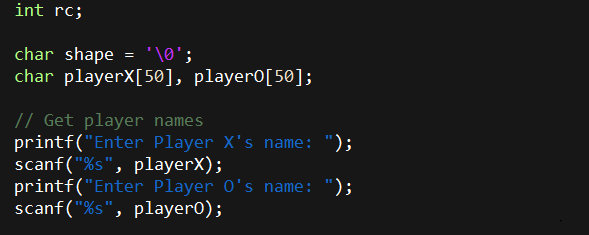
* The game ends when a player wins by aligning 4 consecutive discs in a row, column, or diagonal, or when the grid is full

**Flowchart:**

1. **Start**  
    → Prompt for Player X's name.  
    → Prompt for Player O's name.  
    → Prompt for **number of rows** and **number of columns**.  
    → Allocate memory for the grid (rows x columns).
2. **Game Loop (Turns)**  
    → **Player's Turn**: Player selects a column (within the valid range of columns).  
    → **Check Column Validity**: Is the column full?
   1. Yes: Prompt player to choose another column.
   2. No: Move to the next step.
3. **Place the Disc**  
    → Find the first available space in the chosen column (bottom to top).  
    → Place the player's disc in the grid.
4. **Print Grid**  
    → Display the current state of the grid.
5. **Check for Win**  
    → **Win Check**: Search for 4 consecutive matching discs (horizontal, vertical, diagonal).
   1. If win found: Announce the winner and proceed to **End Game**.
   2. If no win, proceed to the next player.
6. **Switch Turns**  
    → Toggle the turn between Player X and Player O.
7. **End Game**  
    → If a player wins, update the score.  
    → Display current scores from the file.
8. **Exit**

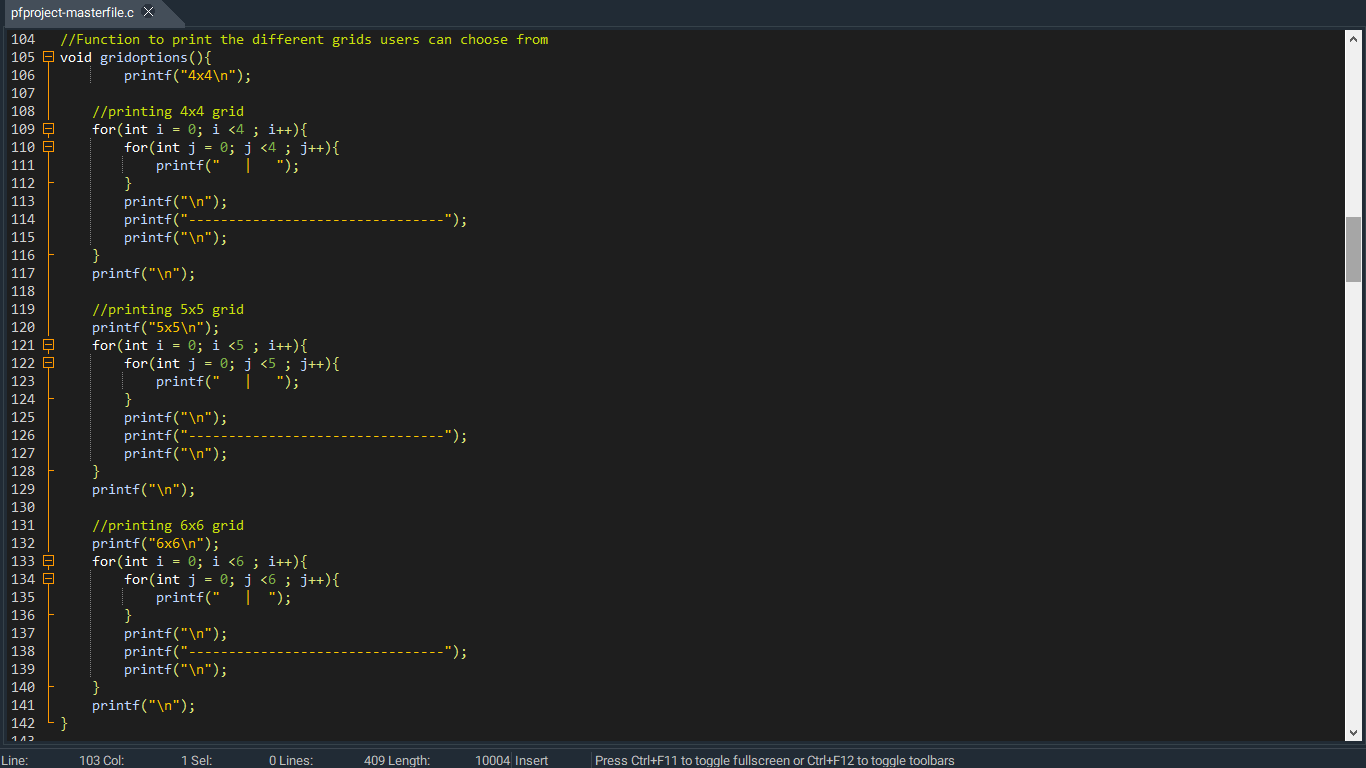
# **Development of the game**

**Inputting player details**



**Grid options function**

The grid options function display grid of size 4x4, 5x5 and 6x6. The user can choose any of the grid size they’d like to play on.



**Dynamic memory allocation**

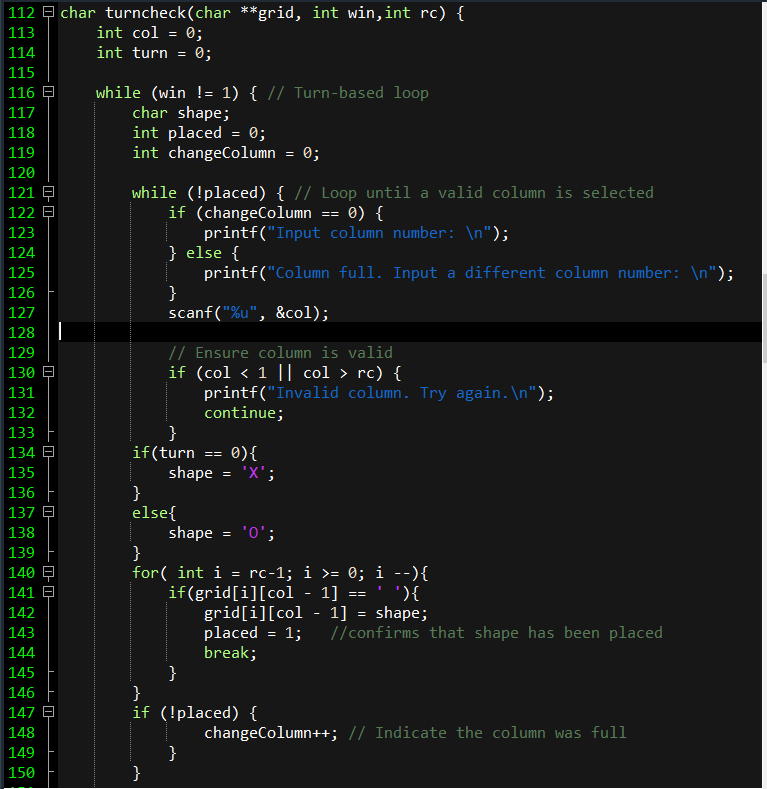
Based on the user input memory is dynamically allocated for the grid.

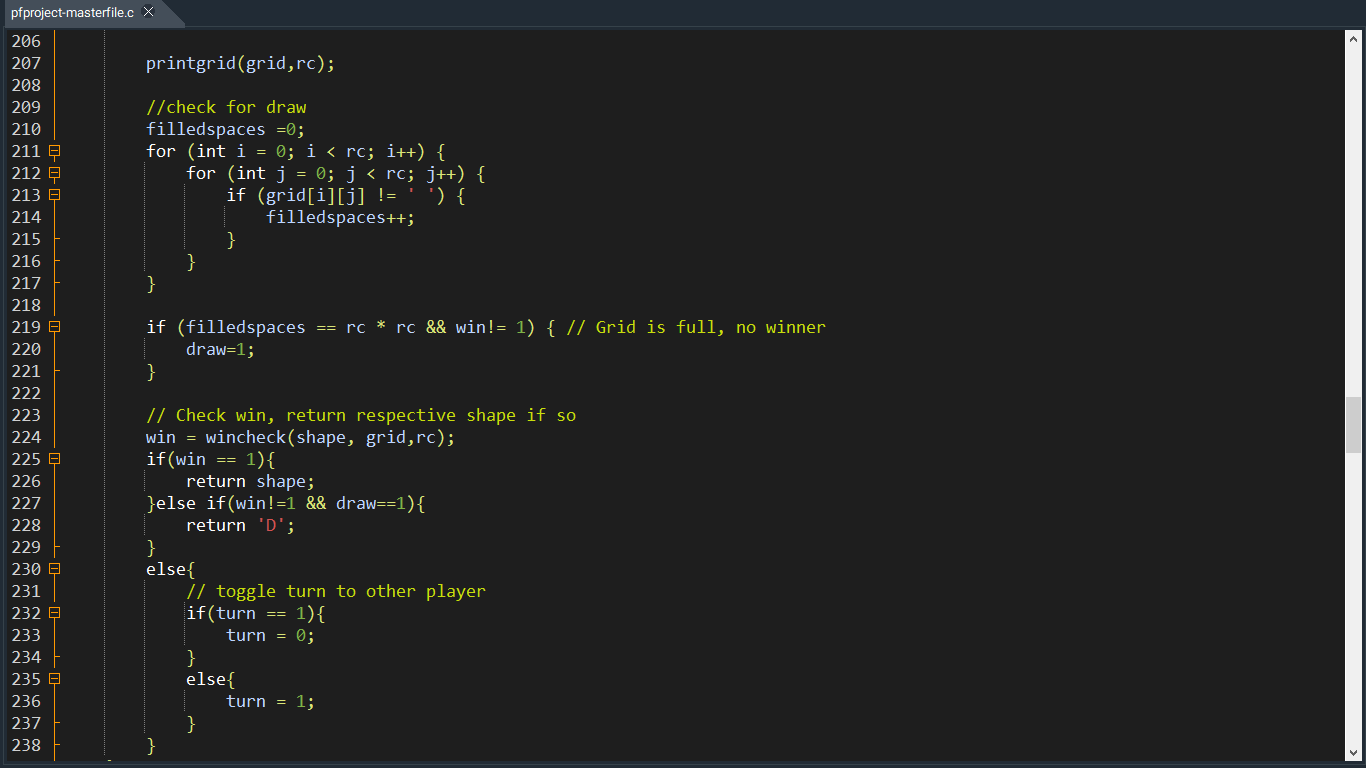
The grid is printed repetitively and the user is prompted to do their turn until the player wins

**Turn check():**

The turn check function prompts the user to enter the column. If it is a valid column program check for the lowest available space in that column. It starts checking from last rows and assign the shape as soon as a free space is found.

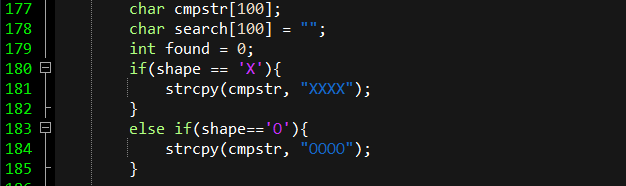
If the column is full user is again prompted to enter the column. After each move program checks for a win. If there is a win the shape of the last turn is returned. Otherwise, the turns are switched and the turn checking process repeats



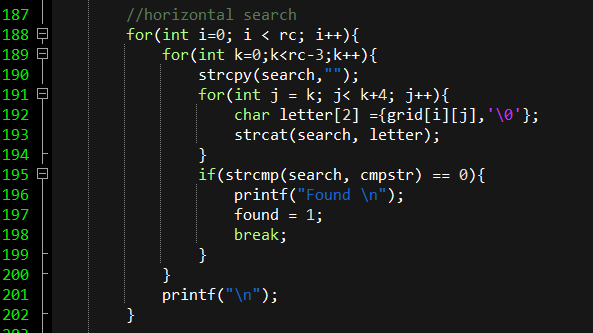


**Wincheck function()**

The wincheck function checks if a player has won in a 2D game grid by finding four consecutive identical symbols ('X' or 'O') either horizontally, vertically, or diagonally

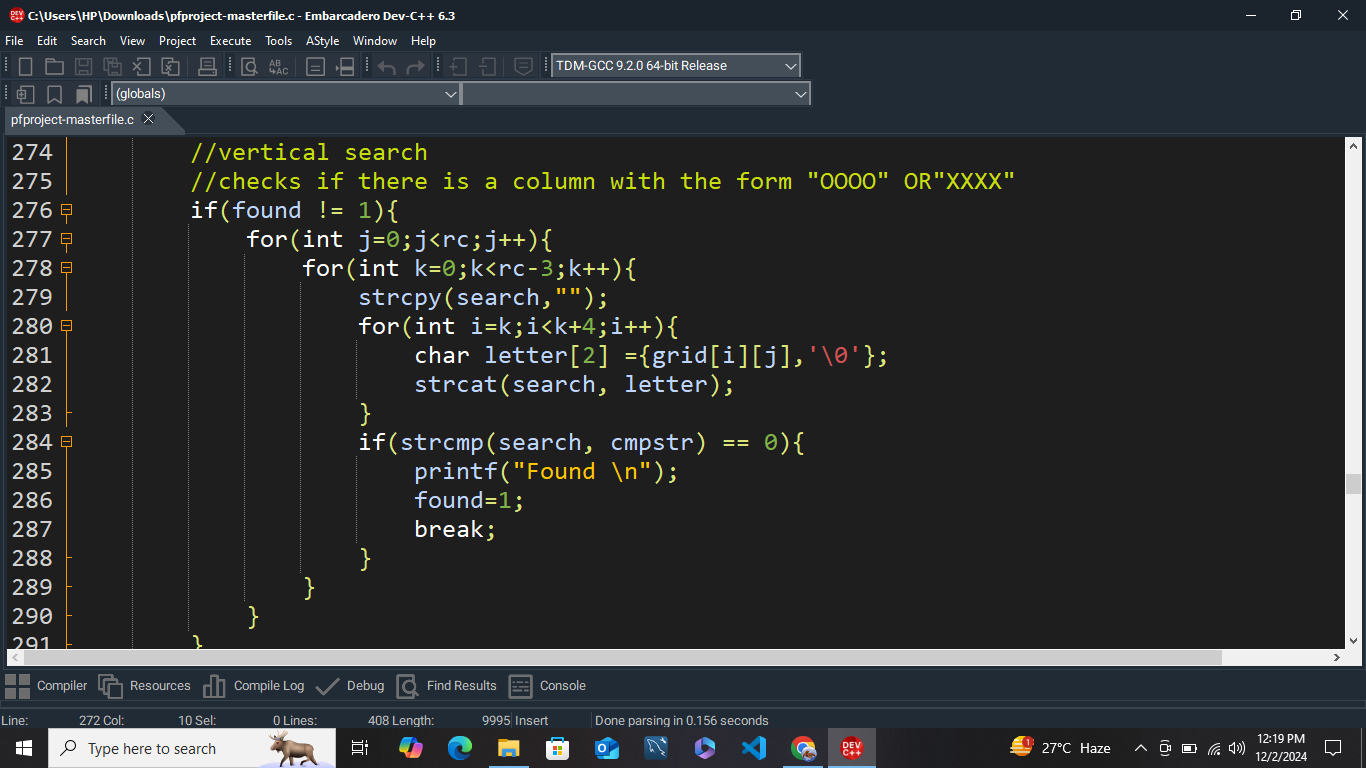
**Horizontal search**

The function iterates through each row and checks all possible 4-cell horizontal sequences by extracting substrings and comparing them with the target string ("XXXX" or "OOOO"). If a match is found, it declares a win.



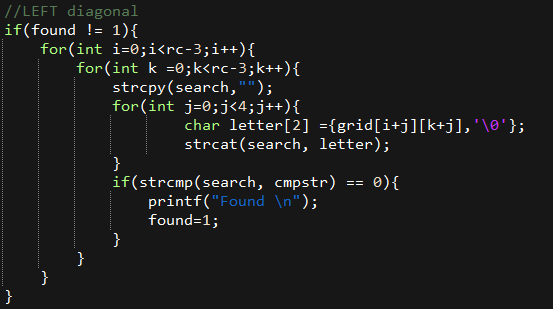
**Vertical search**

Each column is scanned for 4 consecutive identical symbols. Substrings are formed vertically from top to bottom, compared to the target string, and a win is declared if a match is found.



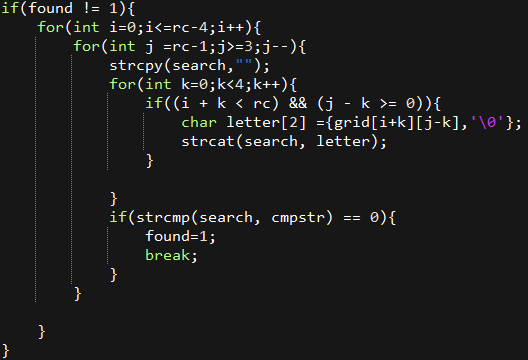
**Left diagonal search**

Starting from the upper-left portion of the grid, the function examines diagonals running from top-left to bottom-right by forming and checking 4-cell sequences against the target string.

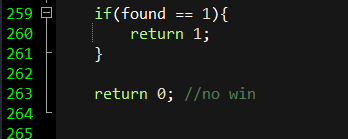


**Right Diagonal search**

From the upper-right portion of the grid, the function evaluates diagonals running from top-right to bottom-left by extracting 4-cell sequences and comparing them to the target string.

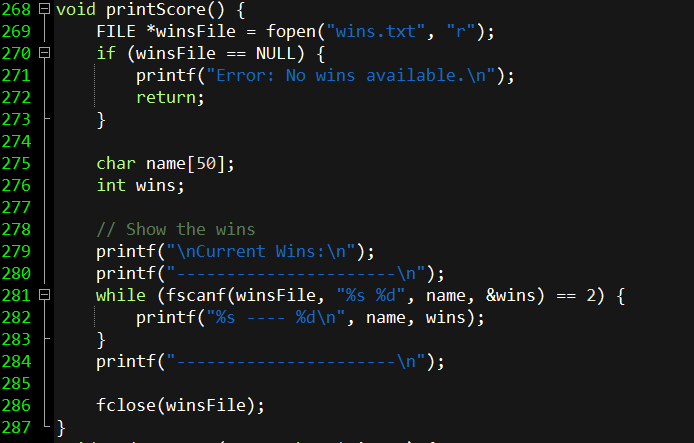


If any of the search found 4 consecutive symbols in a grid 1 is returned from the function indicating there is a win



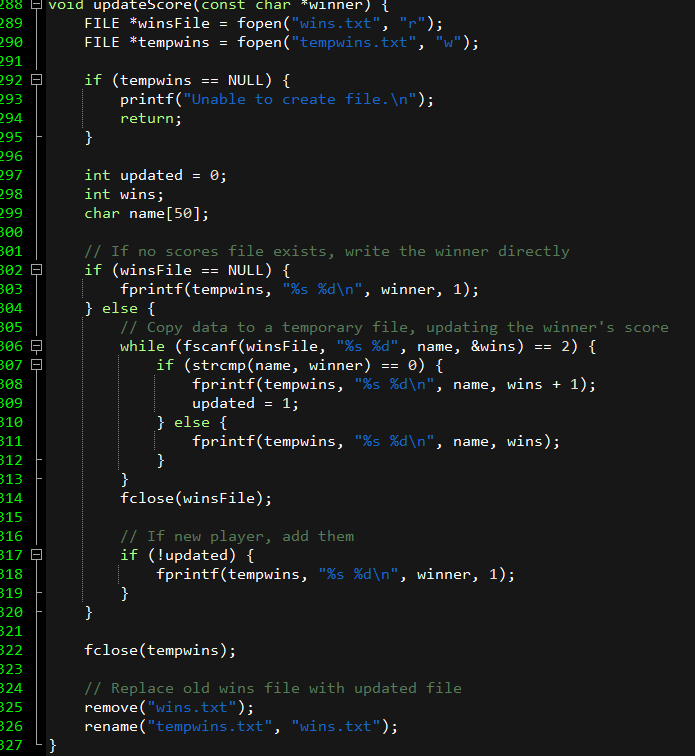
**Printscore()**

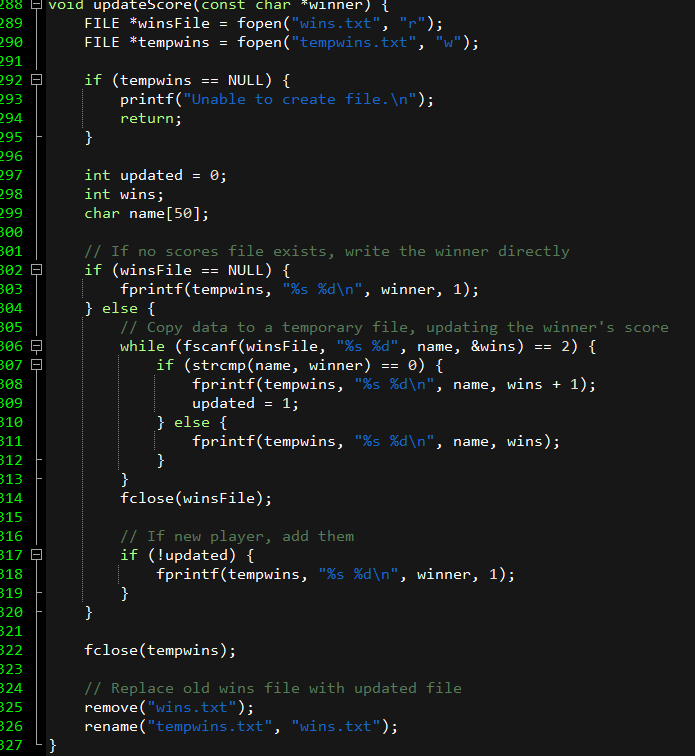
The printScore function reads player names and win counts from a file called "wins.txt" and displays them in a formatted list. If the file cannot be opened, it shows an error message. It uses a loop to read each player's name and wins, printing them with a header and separator. Finally, it closes the file to free resources.



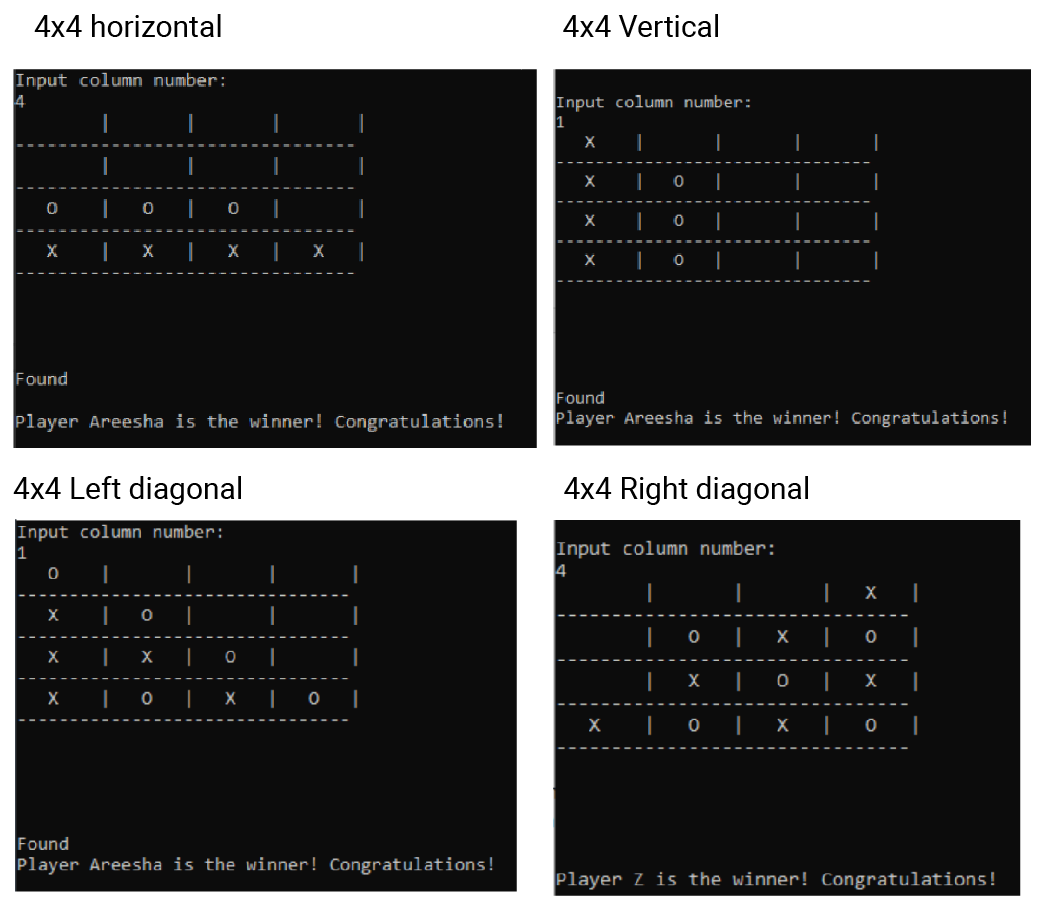
**Update score**

The updateScore function updates the win count for a given player in the "wins.txt" file. If the file exists, it copies its contents to a temporary file while updating the winner's score or adding the winner if they are new. If the file doesn’t exist, it creates a new file with the winner's name and an initial score of 1. After processing, the temporary file replaces the original file.

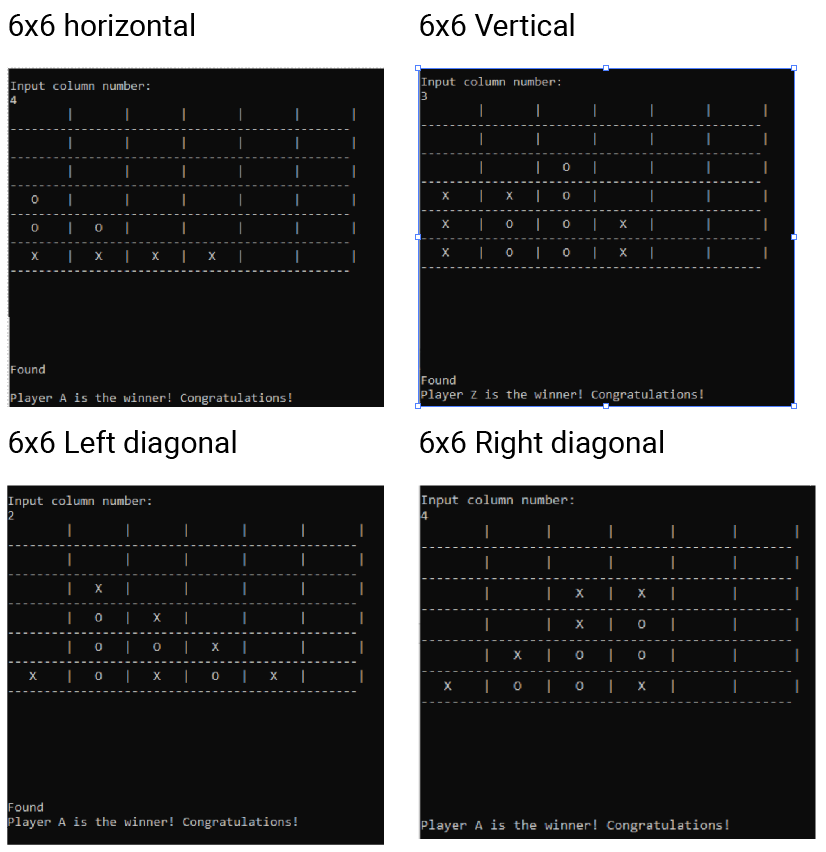




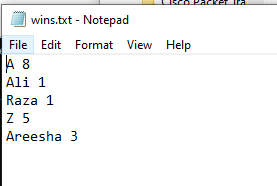
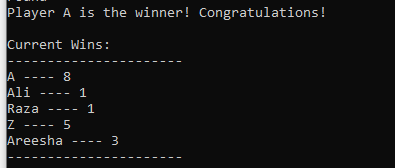
# **Testing**

4x4: 

5x5: 

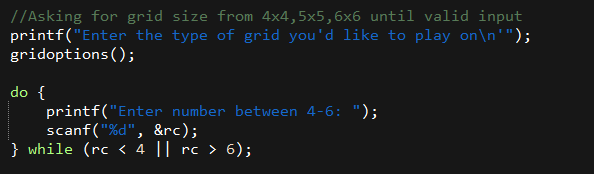
6x6: 

Filing

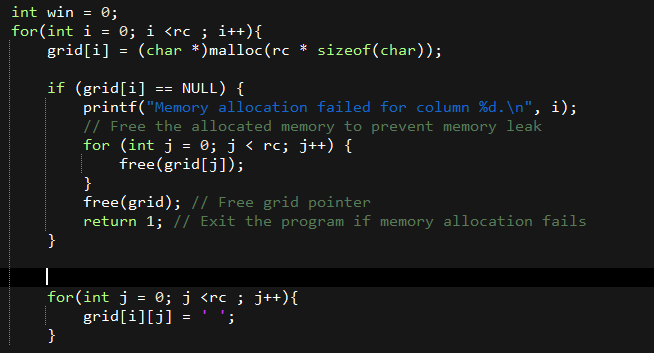


## **Checking for edge cases**

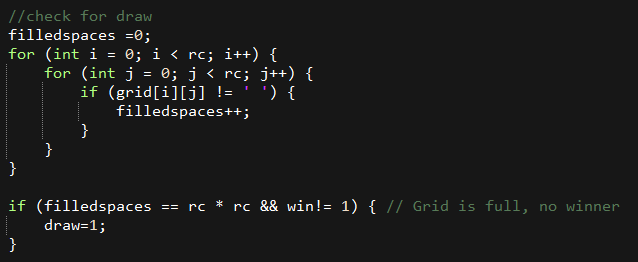
Making sure user only inputs a valid grid option between 4-6



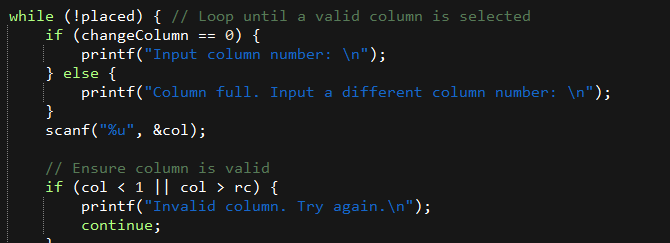
After DMA there are chances that the pointer is empty. We have added checks to ensure the pointer is not NULL



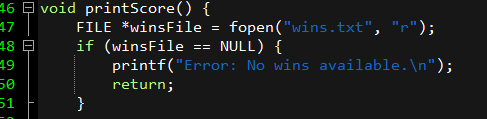
If no one wins and the grid is filled up this leads to the draw situation. If all the slots fill up and there is no win then the filled spaces variable equals available spaces.

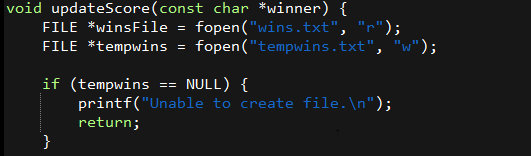


If the column is filled and user tries to input in the same column the user is prompted to enter column name again



We ensured that during filing the pointer is not null. If it is null further tasks are not done on it.

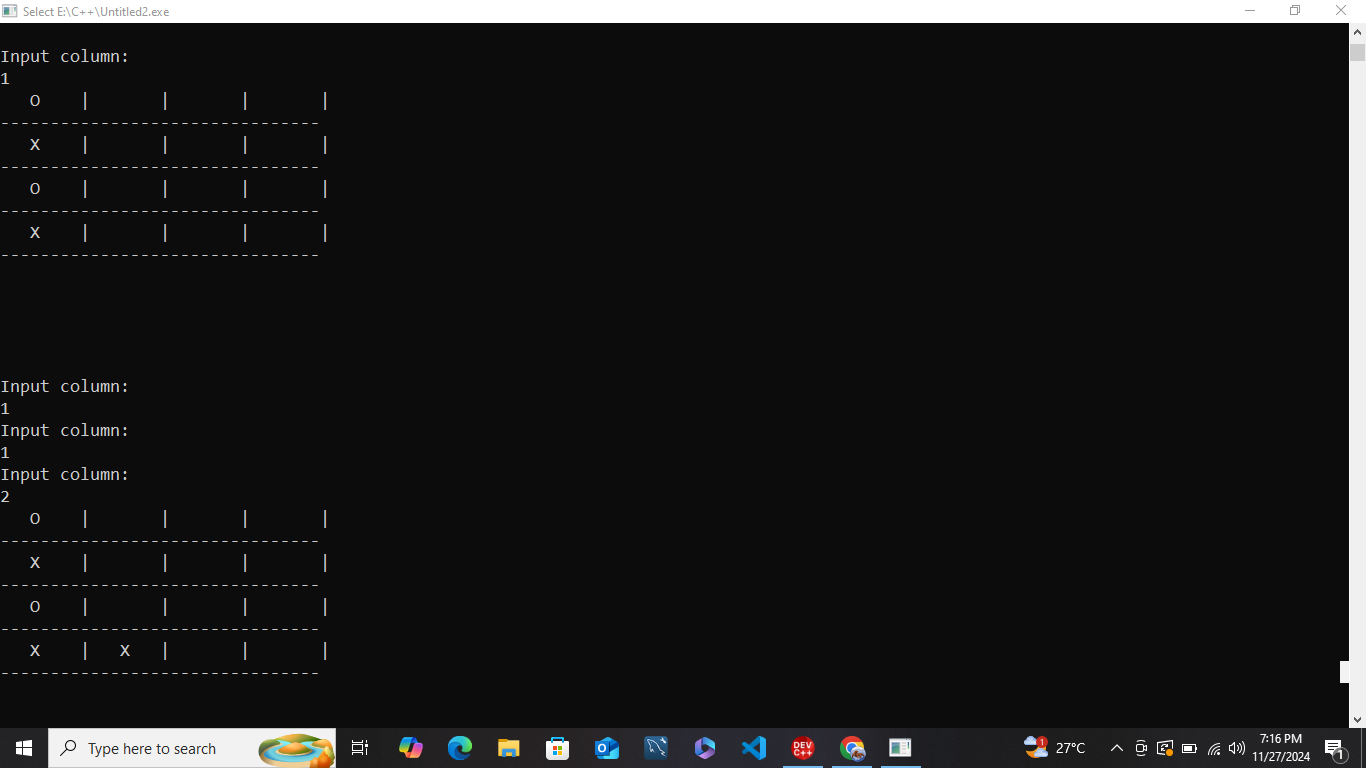




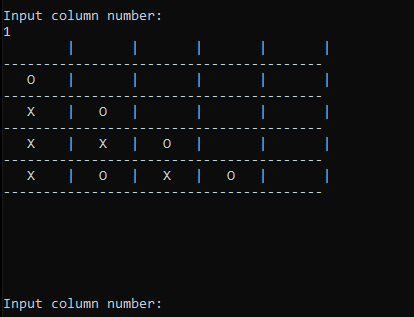
# **Challenges faced**

DMA: Initially we designed the game with a fixed grid size and the loops were hardcoded and the logic was only applicable on a grid of 4x4. Later when we allowed for multiple grid options, the code wasn’t working and some of the logic had to be revised.

Turncheck issue: The previous turncheck didn’t check when columns got filled up. It didn’t indicate that the user needs to input their shape into a separate column. New turncheck function deals with issue.



Placement of setting search string to null: The search string concatenates the characters and compares them to “XXXX” or “OOOO”. The incorrect placement meant that the user was still asked to input a column even when a win condition was met. We had to carefully dry run the code to figure out the issue.



# **Conclusion**

The Connect Four game project successfully demonstrates the application of core programming concepts such as dynamic memory allocation, modular functions, and file handling. Through iterative development, challenges like dynamic grid resizing, turn validation, and efficient win-checking logic were overcome, enhancing both technical skills and problem-solving abilities. The final product delivers a user-friendly, engaging gameplay experience while maintaining robust error handling and code organization.