XO GAMES

A2-Task3-S5-20231116-20231134-20231209

DECEMBER 12, 2024

Illustrative table

Name	ID	What he made
Amr Khaled Ahmed Abd El-Hamid 20231116@stud.fci-cu.edu.eg	20231116	1. Pyramid Tic Tac Toe 1.1. Pyramid_board: 1.1.1. is_within_bounds: bool 1.1.2. check_line: bool 1.1.3. update_board: bool 1.1.4. display_board: void 1.1.5. is_win: bool 1.1.6. is_draw: bool 1.1.7. game_is_over: bool 1.2. PyramidPlayer: 1.2.1. Getmove: void 1.3. PyramidRandomPlayer: 1.3.1. Getmove: void 1.4. PyramidGamemenu: void 2. Word Tic Tac Toe 2.1. Word_Tic_Tac_Toe_Board: 2.1.1. is_within_bounds: bool 2.1.2. check_line: bool 2.1.3. update_board: bool 2.1.4. display_board: void 2.1.5. is_win: bool 2.1.6. is_draw: bool 2.1.7. game_is_over: bool 2.2. Word_Tic_Tac_Toe_Player: 2.2.1. Getmove: void 2.3. Word_Tic_Tac_Toe_Random_Player: 2.3.1. Getmove: void 2.4. PyramidGamemenu: void 3. GUI: 3.1. Building shape of boards 3.2. Accessing symbols on boards

Illustrative table

Name	ID	What he made
Mohamed Ahmed Mohamed Abd El Wahab 20231134@stud.fci-cu.edu.eg	20231134	1. Four-in-a-row 1.1. Four_in_a_row_Board: 1.1.1. update_board: bool 1.1.2. display_board: void 1.1.3. is_win: bool 1.1.4. is_draw: bool 1.1.5. game_is_over: bool 1.2. Four_in_a_row_player: 1.2.1. Getmove: void 1.3. Random_Four_in_a_row: 1.3.1. Getmove: void 1.4. Four_in_a_row_menu: *int* 2. Numerical Tic Tac Toe 2.1. Numerical_Tic_Tac_Toe_Board: 2.1.1. update_board: bool 2.1.2. display_board: void 2.1.3. is_win: bool 2.1.4. is_draw: bool 2.1.5. game_is_over: bool 2.2. Numerical_Tic_Tac_Toe_player: 2.2.1. Getmove: void 2.3. Random_Numerical_Tic_Tac_Toe: 2.3.1. Getmove: void 2.4. Numerical_menu: int 3. Ultimate Tic Tac Toe: 3.1. Ultimate_Tic_Tac_Toe_Board: 3.1.1. update_board: bool 3.1.2. display_board: void 3.1.3. is_win: bool 3.1.4. is_draw: bool 3.1.5. game_is_over: bool 3.2. Ultimate_Tic_Tac_Toe_player: 3.2.1. Getmove: void 3.3. Random_Ultimate_Tic_Tac_Toe: 3.3.1. Getmove: void 3.4. Ultimate_Tic_Tac_Toe_menu: int

Illustrative table

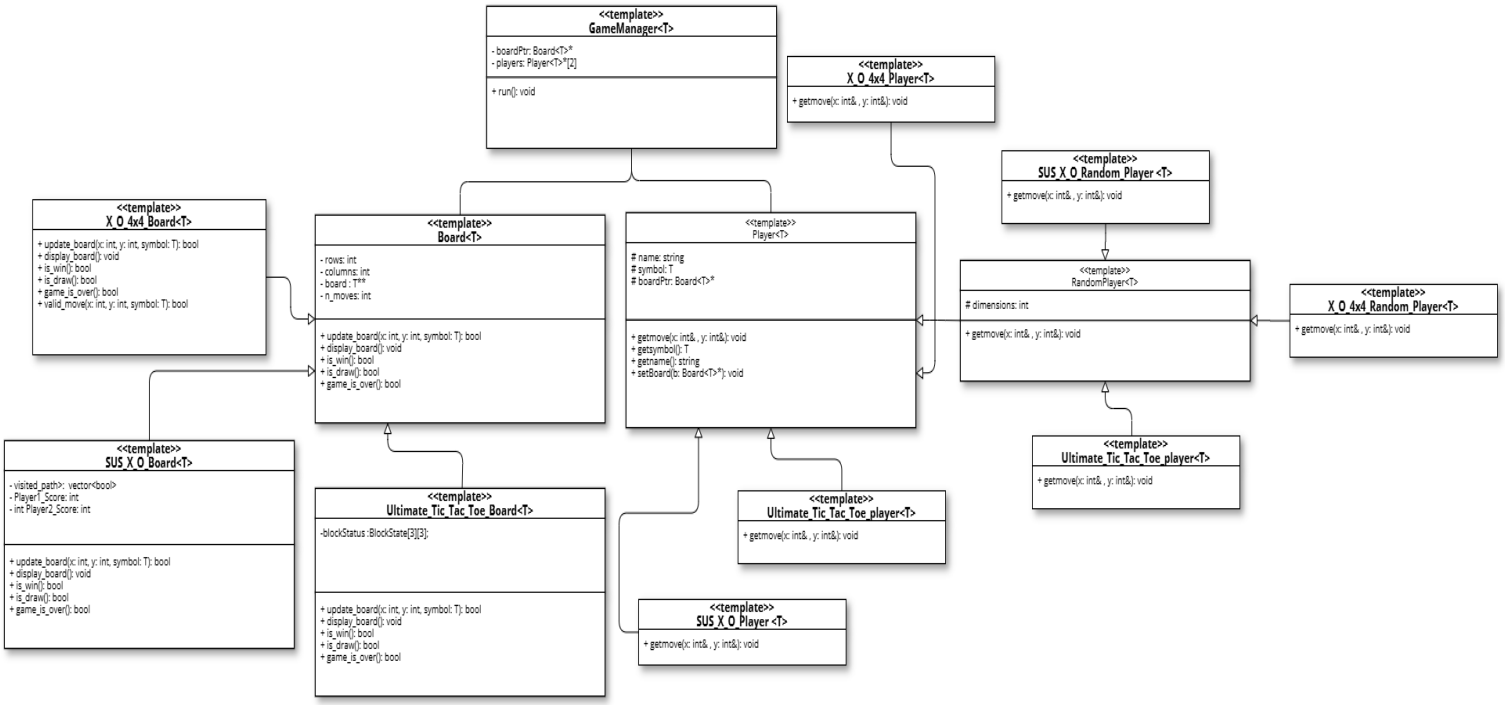
Name	ID	What he made
Youssef Hassan Abd El Gawad yousefhassan8902@gmail.com	20231209	1. 5 x 5 Tic Tac Toe 1.1. X_O_5x5_Board: 1.1.1. update_board: bool 1.1.2. display_board: void 1.1.3. is_win: bool 1.1.4. is_draw: bool 1.1.5. game_is_over: bool 1.2. X_O_5x5_Player: 1.2.1. Getmove: void 1.3. X_O_5x5_Random_Player: 1.3.1. Getmove: void 2. Misere_X_O 2.1. Misere_X_O_Board: 2.1.1. update_board: bool 2.1.2. display_board: void 2.1.3. is_win: bool 2.1.4. is_draw: bool 2.1.5. game_is_over: bool 2.2. Misere_X_O_Player : 2.2.1. Getmove: void 2.3. Misere_X_O_Random_Player: 2.3.1. Getmove: void 3. X_O_4x4: 3.1. X_O_4x4_Board: 3.1.1. update_board: bool 3.1.2. display_board: void 3.1.3. is_win: bool 3.1.4. is_draw: bool 3.1.5. game_is_over: bool 3.2. X_O_4x4_Player: 3.2.1. Getmove: void 3.3. X_O_4x4_Random_Player: 3.3.1. Getmove: void 4. SUS: 4.1. SUS_X_O_Board: 4.1.1. update_board: bool 4.1.2. display_board: void 4.1.3. is_win: bool

		4.1.4. is_draw: bool 4.1.5. game_is_over: bool 4.2. SUS_X_O_Player: 4.2.1. Getmove: void 4.3. SUS_X_O_Random_Player: 4.3.1. Getmove: void
Team Works Together on:		1. Integrating Games on One Menu. 2. Making GUI for 2 Grouped Games. 3. Making Report for All games.

Contents

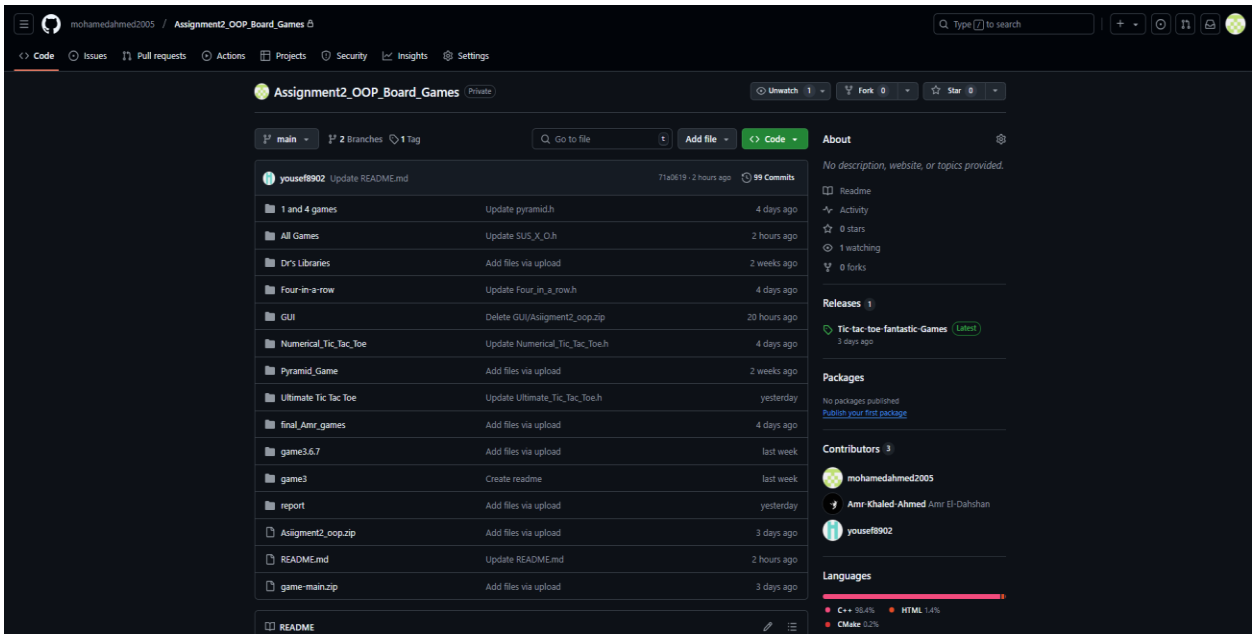
Tic tac toe Board Games

1. UML Diagram:



GitHub repository:

GitHub link: https://github.com/mohamedahmed2005/Assignment2_OOP_Board_Games.git



2. Games:

2.1. Pyramid Tic Tac Toe:

- The game board is shaped like a pyramid. Five squares make the base, then three, then one. Players
- take turns marking Xs and Os as in traditional tic-tac-toe.
- Winning: The first player to get three-in-a-row vertically, horizontally, or diagonally wins.

2.2. Four-in-a-row:

- The game board consists of a 7 x 6 grid. Seven columns of six squares each. Instead of dropping counters as in Connect Four, players mark the grid with Xs and Os as in tic-tac-toe.
- Rules: The first player places an X in the bottom square of any column. Taking turns, players make their mark in any column, as long as it is in the lowest square possible.
- Winning: The first player to get four-in-a-row vertically, horizontally, or diagonally wins.

2.3. 5 x 5 Tic Tac Toe:

- This tic-tac-toe variation is played on a 5 x 5 grid. As in the traditional game, players are Xs or Os.
- Rules: Players take turns placing an X or an O in one of the squares until all the squares except one are filled. (Each player has 12 turns for a total of 24 squares.)
- Winning: Count the number of three-in-a-rows each player has. Sequences can be vertically, horizontally, or diagonally. Whoever has the most, wins.

2.4. Word Tic Tac Toe:

- Word Tic-tac-toe is an innovative twist on the classic Tic-tac-toe game. Instead of using "X" or "O", players place letters on a 3x3 grid to form valid words.
- Players aim to form a valid word with the letters they place on the board. Words can be formed horizontally, vertically, or diagonally.
- Rules: Each player takes turns placing one letter on the board. A player must try to form a valid word with each move. Players can build upon existing letters to form words, provided that the resulting sequence is a valid word.
- Winning: The game is won by forming a valid word horizontally, vertically, or diagonally. If the board fills up without a valid word being formed, the game ends in a draw.

2.5. Numerical Tic Tac Toe:

- Numerical Tic-Tac-Toe offers a mathematical twist to the classic Tic-Tac-Toe game. Instead of the traditional "X" and "O", players use numbers to add an element of strategic calculation.
- The objective is to achieve a sum of 15 with three numbers in a row, column, or diagonal.
- Rules: Player 1 typically starts and uses odd numbers (1, 3, 5, 7, 9), while Player 2 uses even numbers (2, 4, 6, 8). Players alternate turns, placing one number in an empty cell on the board. Each number can only be used once.
- Winning: A player wins by placing three numbers in a row, column, or diagonal that add up to 15.

2.6. Misere Tic Tac Toe:

- Misere Tic Tac Toe, also known as Inverse Tic Tac Toe or Toe Tac Tic, is a unique twist on the classic game. In this version, the objective is to avoid getting three marks in a row. The game flips the traditional win condition on its head, making every move a strategic decision to prevent losing.
- Rules: The game is played on a standard 3x3 Tic-Tac-Toe grid. The goal is to avoid placing three of your marks in a row, column, or diagonal. The player who ends up with three marks in a row loses the game. If all cells are filled without either player aligning three marks in a row, the game ends in a draw.

2.7. 4x4 XO:

- 4 x 4 Tic-Tac-Toe is an extended version of the classic game, played on a larger board with more strategies. Each player has four tokens and aims to align three tokens in a row to win.
- This game introduces new movement rules and strategic depth to the traditional Tic-Tac- Toe.
- The game is played on a 4x4 grid. Each player has four tokens. Tokens are placed in specific starting positions: two tokens on opposite sides of the board for each player.
- Rules: Players alternate turns, moving one of their tokens to an immediately adjacent open square. Tokens can be moved horizontally or vertically but not diagonally. Tokens may not jump over other tokens.
- The goal is to align three of your tokens in a row. This can be achieved horizontally, vertically, or diagonally.
- Winning: The first player to get three tokens in a row wins the game. The alignment can be in any direction: horizontal, vertical, or diagonal.

2.8. Ultimate Tic Tac Toe:

- Ultimate Tic Tac Toe is an expansion of the classic game, where players engage in a meta- game of Tic Tac Toe within a 3x3 grid of smaller Tic Tac Toe boards. The goal is to win three smaller games in a row to claim victory on the main board, adding layers of complexity to the traditional game. The game is played on a large 3x3 grid, where each cell contains a smaller 3x3 Tic Tac Toe board.
- Rules: Player 1 starts by choosing any of the nine smaller Tic Tac Toe boards to play on. Players alternate turns, playing Tic Tac Toe on the smaller boards. The winner of each smaller board claims that space on the main board. The winner of the smaller board replaces that board with their symbol (X or O) on the main board.
- Winning: The first player to win three smaller boards in a row on the main 3x3 grid wins the Ultimate Tic Tac Toe game. The winning row on the main board can be horizontal, vertical, or diagonal.

2.9. SUS Game:

- The SUS game is a simple game played on a 3x3 grid. The objective is to form the sequence "S-U-S" by placing letters in the grid. Players must carefully plan their moves to create as many SUS sequences as possible while blocking their opponent from doing the same.
- Rules: The goal is to create the sequence "S-U-S" in a straight line, which can be achieved diagonally, horizontally, or vertically. Players take turns placing either an "S" or a "U" in any empty square on the grid. A player must use the same letter for each turn. If a player successfully creates an "S-U-S" sequence, they take a point.
- Winning: The game continues until all squares are filled or no more "S-U-S" sequences can be created. The player who creates the most "S-U-S" sequences wins the game.

3. Classes:

3.1. Base Classes:

- **BoardGame_Classes:**
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- **Player:**
  Getmove: move and check if player enters valid move or not.
  Get symbol: to return symbol that player uses it to play.
  Get name: return name of player.
  set board: Set board to play.

- **RandomPlayer:**
  Getmove: move and check if player enters valid move or not.

- GameManager:
  run: to run any game with its instructions.

3.2. Child Classes:

3.2.1. Pyramid Game:

- PyramidBoard:
  is_within_bounds: check if player enter on bounds of board or not.
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- PyramidPlayer:
  Getmove: move and check if player enters valid move or not.

- PyramidRandomPlayer:
  Getmove: move and check if player enters valid move or not.

3.2.2. **Four In a Row Game**:

- Four_in_a_row_Board:
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- Four_in_a_row_player:
  Getmove: move and check if player enters valid move or not.

- Random_Four_in_a_row:
  Getmove: move and check if player enters valid move or not.

3.2.3. 5 x 5 Tic Tac Toe:

- X_O_5x5_Board:
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- X_O_5x5_Player:
  Getmove: move and check if player enters valid move or not.

- X_O_5x5_Random_Player:
  Getmove: move and check if player enters valid move or not.

3.2.4. Word Tic Tac Toe Game:

- Word_Tic_Tac_Toe_Board:

  Set “words”: to store words from text file
  is_within_bounds: check if player enter on bounds of board or not.
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- Word_Tic_Tac_Toe_Player:
  Getmove: move and check if player enters valid move or not.

- Word_Tic_Tac_Toe_Random_Player:
  Getmove: move and check if player enters valid move or not.

3.2.5. Numerical Tic Tac Toe Game:

- Numerical_Tic_Tac_Toe_Board:
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- Numerical_Tic_Tac_Toe_player:
  Getmove: move and check if player enters valid move or not.

- Random_Numerical_Tic_Tac_Toe:
  Vector “ available_numbers”: to store available numbers that user can choose.
  Getmove: move and check if player enters valid move or not.

3.3.6. Misere Tic Tac Toe Game:

- Misere_X_O_Board:
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- Misere_X_O_Player:
  Getmove: move and check if player enters valid move or not.

- Misere_X_O_Random_Player:
  Getmove: move and check if player enters valid move or not.

3.3.7. 4X4 Tic Tac Toe Game:

- X_O_4x4_Board:
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- X_O_4x4_Player:
  Getmove: move and check if player enters valid move or not.
- X_O_4x4_Random_Player:
  Getmove: move and check if player enters valid move or not.

3.3.8. Ultimate Tic Tac Toe Game:

- Ultimate_Tic_Tac_Toe_Board;
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- Ultimate_Tic_Tac_Toe_player:
  Getmove: move and check if player enters valid move or not.

- Random_Ultimate_Tic_Tac_Toe:
  Getmove: move and check if player enters valid move or not.

3.3.9. **SUS Tic Tac Toe Game:**

- SUS_X_O_Board:
  vector<bool> visited_path: to check if this path taken or not
  int Player1_Score: to count number that first player wins
  int Player2_Score: to count number that second player wins
  Update Board: to update board using symbol of each player.
  display board: this function to display board on terminal.
  Is win: to check if player wins or not
  Is draw: to check if board is full and no player wins
  game_is_over: check if there is winner or draw

- SUS_X_O_Player:
  Getmove: move and check if player enters valid move or not.

- SUS_X_O_Random_Player:
  Getmove: move and check if player enters valid move or not.

3. Code Quality:

3.1. Pyramid Game:

- Missing headers: The code is missing necessary headers for including libraries, e.g., #include <ctime>, #include <cstdlib>. These headers are required for specific functionalities like srand() and rand().
- Using standard library inappropriately: The standard namespace std is used but not explicitly included in the right manner. Instead of using namespace std;, it's better to use the prefix std:: where necessary to avoid naming conflicts.
- Memory management: The code lacks proper memory management practices. It allocates memory in the constructor of PyramidBoard but does not handle exceptions in case of memory allocation failures. Hence, it would be beneficial to incorporate exception handling mechanisms for robust memory allocation.
- To address the issues, the provided code has been updated with the necessary header inclusions, adjusted the usage of the standard namespace by replacing using namespace std; with the appropriate std:: prefix, and improved memory management practices by introducing exception handling for memory allocation.

3.2. Numerical Tic Tac Toe:

- The provided code has some vulnerabilities and syntax issues that have been addressed in the optimized version:
- Added include directives for all the necessary libraries.
- Changed the 'numerical_menu' function to the correct format 'Numerical_menu.'
- Fixed the 'get_move' function inside 'Numerical_Tic_Tac_Toe_player' to ensure valid input handling.
- Ensured proper checking for valid moves and input validation in the 'get_move' function to prevent errors.
- Cleanup of variables and objects after the game finishes to avoid memory leaks.

3.3. Four in a row:

- Missing headers and namespaces for the classes used, like Player, RandomPlayer, and GameManager,
- Retrieving and comparing user choices as strings and failing to handle invalid inputs or out of range conditions properly,
- Maintaining last move info for the random player within the scope of the function, causing it to always make the same move,
- Errors and warnings during compilation due to incorrect ranges or comparisons in various functions.
- To address these issues, the following changes were made:
- Added missing headers for standard libraries and class interfaces,
- Replaced raw string comparison and input validation methods with integer values and loops to ensure valid choices are entered,
- Moved the declaration of last_x and last_y variables to the scope of the class to keep track of random moves correctly,
- Fixed issues like comparing ranges or checking boundaries in different functions.
- After making these modifications, the code now functions correctly and efficiently.

4.4. Word Tic Tac Toe:

- The code had the following issues:
- The use of using namespace std; is avoided to prevent namespace pollution. Each standard library entity is now prefixed with std::.
- Added #include <iostream> to support the use of I/O streams such as std::cout and std::cin.
- Replaced all instances of cerr with std::cerr and endl with std::endl to qualify them with the correct namespace.
- Added the include for <iostream> which was missing but required for I/O operations like displaying messages.

4.5. 4X4 Tic Tac Toe:

- The code provided is a header file containing template classes for a game board (X_O_4x4_Board), player (X_O_4x4_Player), and a random player (X_O_4x4_Random_Plaver) for a 4x4 Tic-Tac-Toe game. Here are the identified issues and solutions:
- **Header Guards:** Added include guards to the header file to prevent multiple inclusions of the same header.
- **Std Namespace:** Replaced using namespace std; with individual std:: where necessary to avoid polluting the global namespace.
- **Include Headers:** Removed duplicate inclusion of <iostream> and added necessary headers for dependencies like <string> and <cstdlib>.
- **Global Variables:** Moved global variables oldX_pos and oldy_pos inside the class scope.
- **Implementation:** The implementation part of the classes has been omitted for brevity but should follow the declarations and includes necessary logic for the game to work effectively.

4.6. Ultimate Tic Tac Toe:

- The updated code fixes the following issues:
- Added missing include directive for "BoardGame_Classes.h" to resolve any potential issues related to this external header file.
- Added necessary 'new' parameters as empty '()' while initializing the dynamic array 'board = new T*[this->rows]()' to avoid potential issues with uninitialized memory.

4.7. 5X5 Tic Tac Toe:

- Missing include directive for string.
- Using string instead of std::string.
- Incorrect comparison (== instead of =).
- Using cout directly without proper namespace (std::cout).
- Missing include for numeric_limits.
- Missing include for time.h.
- To address these issues:
- Added an include for <string>.
- Updated occurrences of string to std::string.
- Corrected the comparison operations.
- Prefixed cout, cin, and other standard library functions with std::.
- Added an include for <limits>.
- Added an include for <ctime> to use the time() function.

4.8. Misere Tic Tac Toe:

- Removed duplicate includes and added missing #include <cstdlib> and #include <ctime> for using rand() and srand() functions respectively.
- Fixed missing std:: namespace for string in Misere_X_O_Player constructor.
- Changed lose and moves variables to be members of the board class, dropped the use of global variables.

4.9. SUS GAME:

- Added missing #include <string> header for using the string data type.
- Added using namespace std; to prevent compilation errors due to missing namespace for elements like vector.
- Added #include <cstdlib> and #include <ctime> for using the rand() function for random number generation.
- Replaced > with < in the condition x < 2 to fix the lower bound check for valid coordinates.
- Initialized symbol variable in ternary conditions to avoid potential uninitialized value usage.
- Fixed logic issues related to choosing and assigning symbols for the players during the game.
- Updated code formatting and indentation for better readability.

5. GUI:

5.1. Video Link:

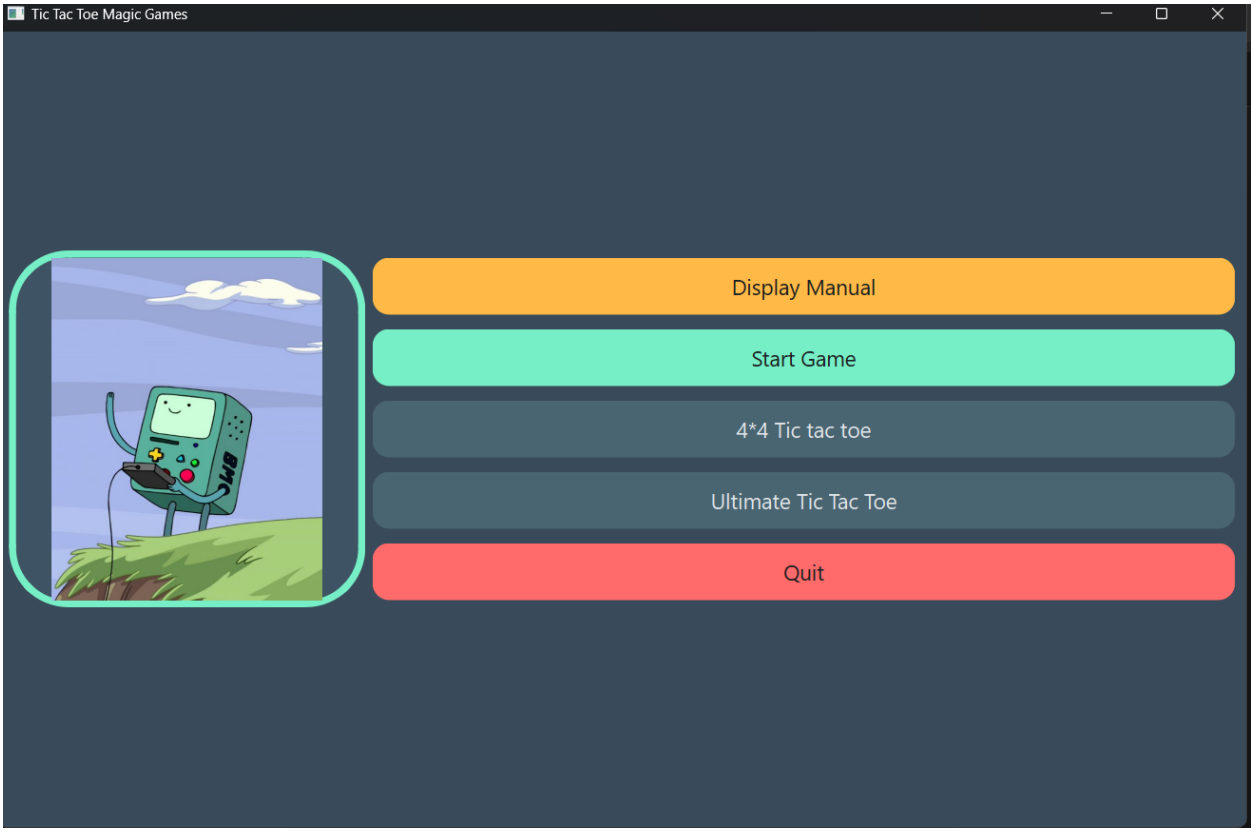
- https://youtu.be/P-ERgVQWDRI?feature=shared

5.2. Github Link:

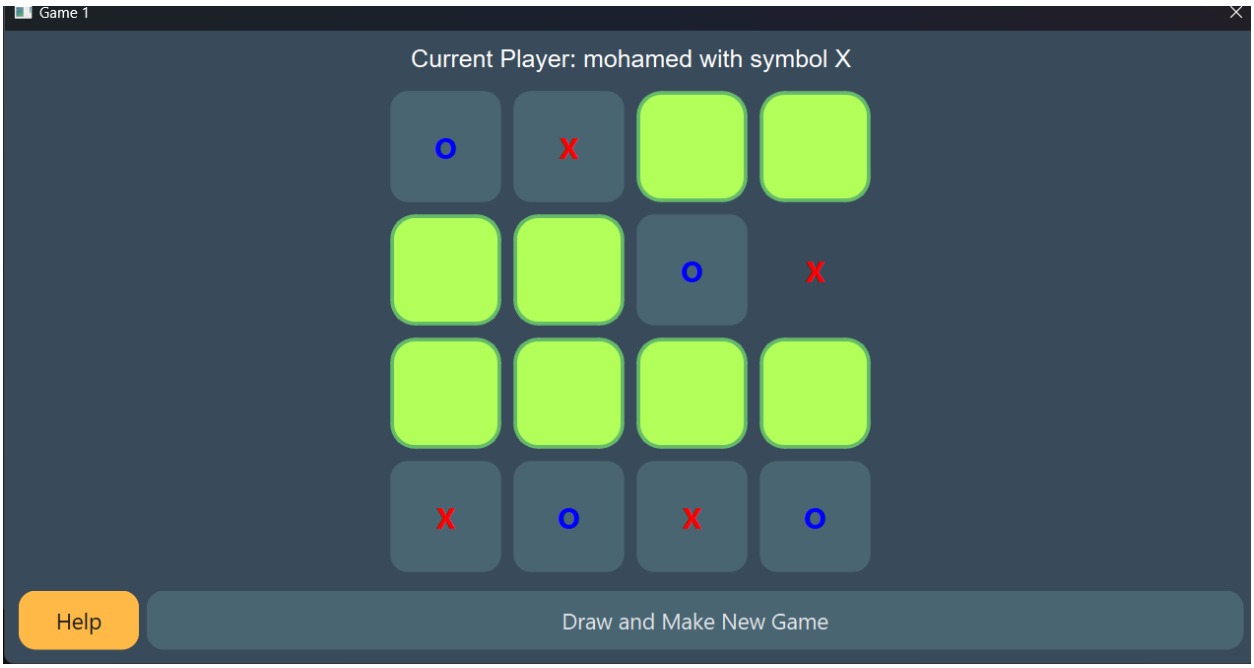
https://github.com/mohamedahmed2005/Assignment2_OOP_Board_Games/releases/tag/Games-V1
https://github.com/mohamedahmed2005/Assignment2_OOP_Board_Games

5.3. Photos From program:

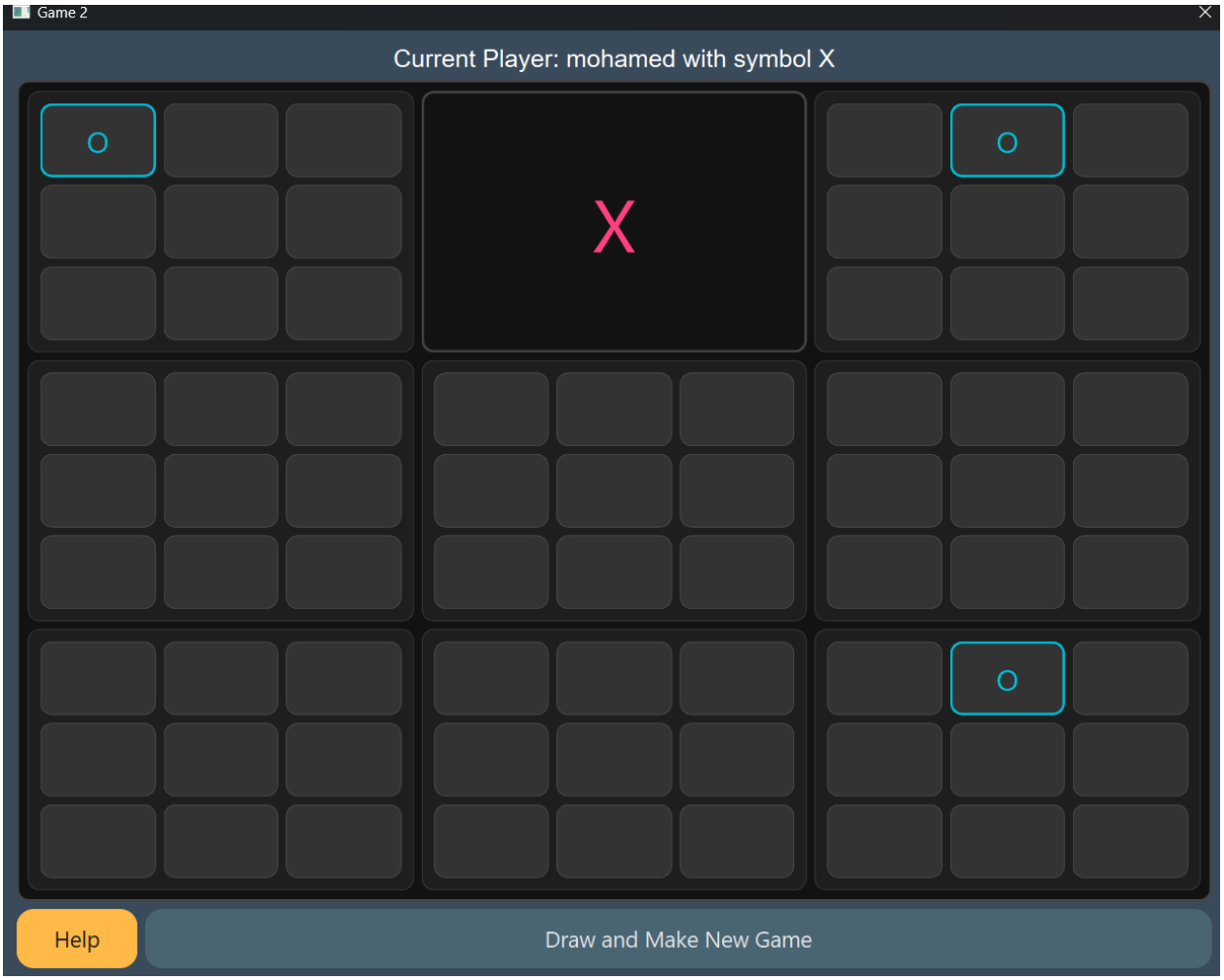
- **Main window:**



- 4*4 Tic tac toe



- Ultimate Tic tac toe



6. Appendix

Appendix A
website that we use on code quality: https://www.clouddefense.ai/tools/code-checker/cpp