Classes

1-Pyramic Tic Tac Toe

The game board is shaped like a pyramid. Five squares make the base, then three, then one.

Rules:

- 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-Four in a Row

A 2D twist on the classic Connect Four game!

Rules:

-The board is a 7x6 grid (7 columns, 6 rows).

-Player 1 uses 'X'; Player 2 uses 'O'.

-Players take turns marking the lowest available cell in a column.

Winning:

Form four in a row:

-Horizontally

-Vertically

-Diagonally

A draw occurs if the board is full with no winners.

3-5x5 Tic Tac Toe

A variation of Tic-Tac-Toe played on a 5x5 grid.

Rules:

-Players take turns placing an X or an O in one of the squares.

-The game ends when all squares except one are filled (each player has 12 turns).

Winning:

-Count the number of three-in-a-row sequences (horizontal, vertical, or diagonal) for each player.

-The player with the most three-in-a-row sequences wins.

-Decide ahead of time whether a mark can be counted in more than one sequence.

4-Word Tic Tac Toe

An innovative twist to the classic game, instead of 'X' and 'O', players place letters on a 3x3 grid. This adds a linguistic challenge to the traditional game mechanics.
Rules:

-Players aim to form a valid word with the letters they place on the board
-Words can be formed horizontally, vertically, or diagonally.

Winning:

-The first player to get three in a row vertically, horizontally, or diagonally wins.

5-Numerical Tic Tac Toe

A mathematical twist on Tic-Tac-Toe! Players use numbers instead of "X" and "O".
Rules:

-Player 1 uses odd numbers (1, 3, 5, 7, 9).
-Player 2 uses even numbers (2, 4, 6, 8).

Objective:

-Achieve a sum of 15 in a row, column, or diagonal.
-Numbers are placed in empty cells and can only be used once.

Winning:

-Form a row, column, or diagonal with a sum of 15 to win.
-If all cells are filled and no sum of 15 is achieved, it's a draw.

6-Misere Tic Tac Toe

A twist on the classic Tic-Tac-Toe where the goal is to avoid getting three marks in a row.
Rules:

-Players take turns placing an X or an O in one of the squares.
-The goal is to avoid forming three 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 forming three marks in a row, the game ends in a draw.

Winning:

-There is no winning condition. Instead, the player who places three marks in a row loses.
-The game ends in a draw if no player forms a row of three marks and all cells are filled.

7-4x4 Tic Tac Toe

An extended version of Tic-Tac-Toe with a 4x4 board.
Rules:

-The board is a 4x4 grid.
-Players start with four tokens in specific positions.
-Tokens move to adjacent open squares (horizontally/vertically).
-Tokens cannot jump over others or move diagonally.

Winning:

-Align three tokens in a row:

-Horizontally
-Vertically
-Diagonally

The game ends when a player achieves this goal.

8-Ultimate Tic Tac Toe

An expanded version of Tic-Tac-Toe with a 3x3 grid of smaller boards.
Rules:

-The main board is a 3x3 grid, each cell containing a smaller 3x3 Tic Tac Toe board.
-Player 1 starts by choosing any of the nine smaller boards to play on.
-Players alternate turns, playing Tic Tac Toe on the smaller boards.
-The winner of each smaller board places their symbol (X or O) on the main board.

Winning:

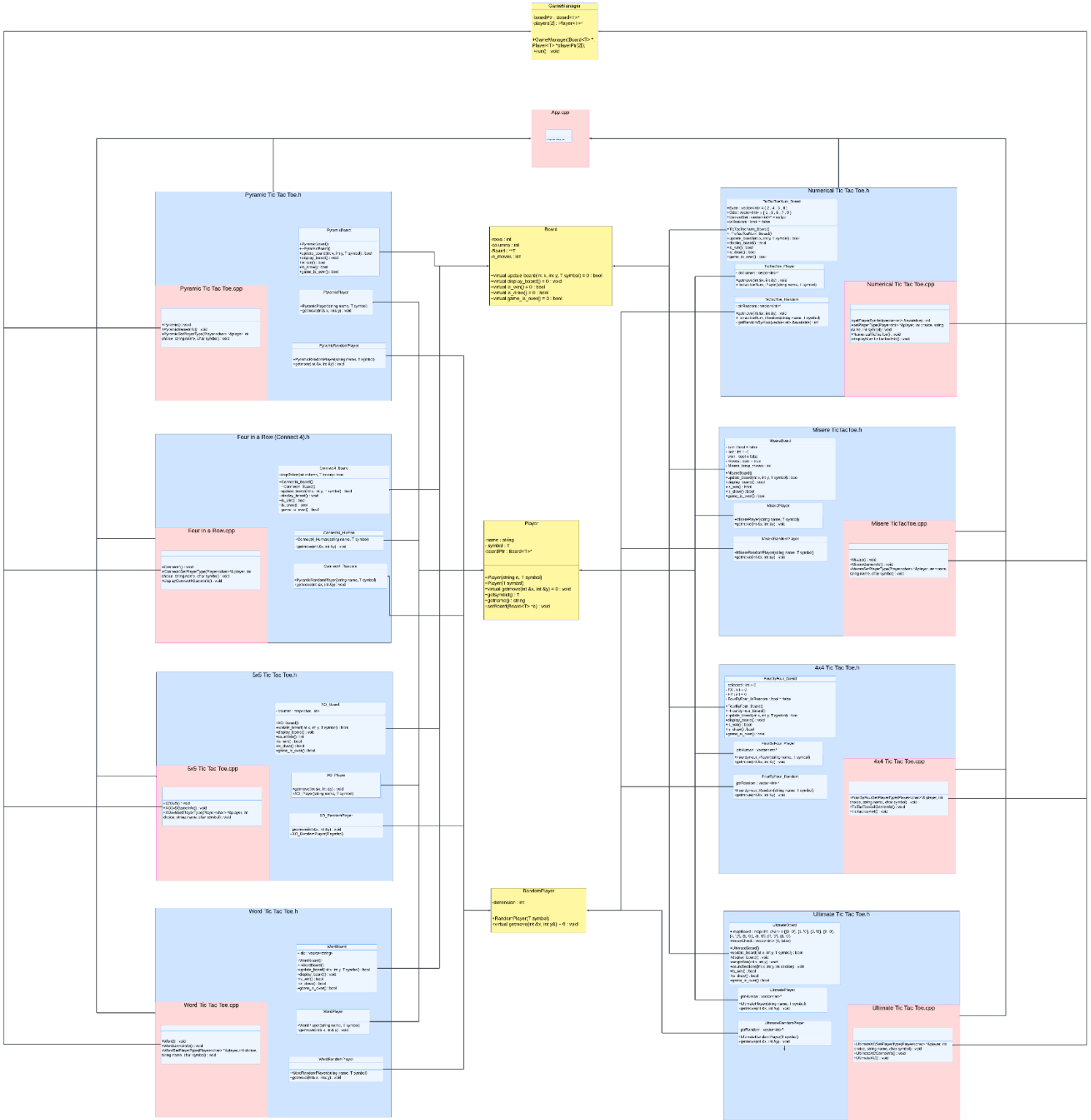
-The first player to win three smaller boards in a row (horizontal, vertical, or diagonal) on the main board wins the game.

Work Breakdown

Author	Game	Task Number
Loai Hataba	5x5 Tic Tac Toe	3
Loai Hataba	Misere Tic Tac Toe	6
Loai Hataba	Ultimate Tic Tac Toe	8
Hossam Abdelaziz	Pyramic Tic Tac Toe	1
Hossam Abdelaziz	Word Tic Tac Toe	4
Abdullah Mohamed	Four in a row	2
Abdullah Mohamed	Numerical Tic Tac Toe	5
Abdullah Mohamed	4x4 Tic Tac Toe	7

UML Design (Link for viewing:

https://drive.google.com/file/d/1ZCCyyB8Yy6b81sC9CfPqtn5UOw7Vnb5y/view?usp=sharing



Reports

Loai:

Pyramid Tic-Tac-Toe Feedback (Hossam's Code)

Hossam’s version of Pyramid Tic-Tac-Toe brings a creative twist to the classic game with its unique triangular board. The game is easy to follow, thanks to a user-friendly display and clever input mapping. The error handling for invalid moves makes the gameplay smoother, and the pyramid design adds an interesting challenge.

However, there are some areas where the code could be improved. The board size is hardcoded, which makes it difficult to change or expand. The random player in the game doesn’t check if a move is valid, which could cause issues or crashes. Some parts of the code, like checking for a win, are repeated unnecessarily, which makes it harder to maintain. Additionally, some error messages feel unprofessional and could be more polished. Improving these aspects would make the game more flexible, reliable, and professional.

Word Tic-Tac-Toe Feedback (Hossam's Code)

Hossam’s Word Tic-Tac-Toe offers a fun twist by combining word-building with traditional gameplay. The game uses a dictionary to check words, and the flexibility to play as either a human or random player is a nice touch. The design is thoughtful, and the gameplay experience feels unique.

There are a few things that could be better. The board size is still hardcoded, which limits how the game can be modified. The way the game checks for a win is a bit slow and could be improved. The code for handling the dictionary isn’t very resilient, which might cause issues, and the error messages could be more professional. Also, the random player doesn’t always play valid moves, leading to inefficiencies. By making the board size adjustable, improving how the game checks for wins, and refining the error handling, Hossam could make the game more polished and enjoyable.

Connect 4 Feedback (Abdullah's Code)

Abdullah’s Connect 4 game is easy to follow and correctly implements the core gameplay, with clear management of the board and players. The game can detect when a player wins, which is a key feature.

However, there are a few things that could make the code better. The board size and some game settings are hardcoded, so it’s difficult to adjust the game for different setups. There isn’t much validation for player input, which could lead to unexpected errors during the game. Also, there’s some repetition in the win-checking logic, which could be streamlined. Finally, the code doesn’t include any logging, which would make it harder to track errors. By making the board size adjustable, adding input checks, and reducing repetition in the win conditions, Abdullah could improve the game's flexibility and reliability.

Four by Four XO Feedback (Abdullah's Code)

Abdullah’s Four by Four XO game offers a nice variation on the classic Tic-Tac-Toe with a larger board. The game structure is clear, and the win-checking feature works well.

There are some areas where the game could be improved. Like with Connect 4, the board size is fixed, which makes the game less flexible. The code doesn’t check if the moves are valid, which could cause issues. Also, there’s no logging for errors, which would help with debugging. The code for checking if a player has won is repeated, which could be simplified. Abdullah should consider making the board size adjustable, adding better input checks, and improving the win-checking logic to make the game more flexible and easier to maintain.

Tic-Tac-Toe Numbers Feedback (Abdullah's Code)

Abdullah’s Tic-Tac-Toe Numbers variant is a fun take on the classic game, using numbers instead of symbols. The game logic works as expected, and it successfully detects when a player wins.

However, there are some improvements to consider. The board size is fixed, which makes it hard to adjust or scale the game. There’s little validation for player input, which could lead to errors during play. The win-checking logic is repeated in multiple places, which could be simplified. Also, there’s no logging to help track or fix errors. Abdullah should think about making the board size flexible, improving input checks, and cleaning up the code for detecting wins to make the game more scalable and reliable.

Abdullah:

Misère Tic-Tac-Toe Feedback

The code effectively implements a Misère version of Tic-Tac-Toe, creatively flipping the objective by making players lose if they form three in a row. It employs a class-based structure, leveraging inheritance to extend functionality for specialized players. The overall approach showcases a sound understanding of object-oriented programming principles and modularity, though improvements are needed to enhance readability, maintainability, and robustness. With refinements, the code could better adhere to clean coding practices and handle edge cases more gracefully.

Strengths include a user-friendly display_board method that makes the game intuitive and engaging. The incorporation of unique Misère-specific logic adds strategic complexity, differentiating it from traditional Tic-Tac-Toe. Additionally, inheritance minimizes code duplication in player-specific classes, promoting reusability and maintainability. However, weaknesses include the use of global state variables, inconsistent naming conventions, and hardcoding of board dimensions, which hinder scalability. Redundancies in win-condition logic and insufficient input validation further detract from the robustness of the implementation.

5x5 Tic-Tac-Toe Feedback

The code provides a creative take on 5x5 Tic-Tac-Toe, incorporating unique rules like counting sequences of three-in-a-row to win. Its object-oriented approach with inheritance for player types and modular board design stands out as a strength, enabling clear separation of concerns. The visually clean and intuitive board display, alongside strategic win-tracking via a countWin function, highlights the effort to enhance gameplay depth and accessibility.

However, notable weaknesses include reliance on global state variables and hardcoded board dimensions, limiting flexibility and introducing maintenance challenges. Input validation is minimal, increasing the risk of errors during gameplay, and redundant logic in the countWin function reduces efficiency. Furthermore, the code lacks proper memory management for dynamically allocated arrays, leading to potential memory leaks. Addressing these issues would elevate the code's quality, making it more scalable and robust.

Pyramid Tic-Tac-Toe Feedback

The Pyramid Tic-Tac-Toe implementation stands out with its innovative triangular board structure and non-standard gameplay. A user-friendly display_board function and custom input mapping through makeIndex improve usability. Error handling for invalid moves ensures smoother gameplay, while the pyramid design introduces a refreshing challenge to the Tic-Tac-Toe formula, demonstrating creativity and thoughtful design.

On the other hand, hardcoded board dimensions and dependency on global functions like makeIndex limit scalability and modularity. The PyramidRandomPlayer fails to validate moves, leading to inefficiencies or potential crashes. Repeated logic for win conditions reduces maintainability, and verbose hardcoded messages like "Invalid ya broooooooo!!!" detract from professionalism. Improving input validation, encapsulating global dependencies, and abstracting duplicate logic would significantly enhance the code's quality.

Word Tic-Tac-Toe Feedback

This Word Tic-Tac-Toe variant creatively combines word-building with traditional gameplay, utilizing a dictionary for validation. Features like dictionary customization and a flexible player implementation (human and random players) add to its appeal. The use of polymorphism and a unique isValidWord function to check win conditions highlight the code's thoughtful design and gameplay innovation.

Despite these strengths, the code is limited by hardcoded board dimensions and inefficient win-checking logic, which complicates scalability. Dictionary handling lacks error resilience, and verbose outputs such as "Invalid ya broooooooo!!!" affect its professionalism. Furthermore, the random player's inability to validate moves creates inefficiencies. Resolving these shortcomings through parameterized board sizes, efficient word validation, and polished outputs would significantly improve the code's robustness and user experience.

Hossam:

Abdallah

Connect-4

Game logic is well implemented by using dropToken function to correctly implement connect4 rules.

Checking for valid input (e.g. Column is full can't drop more, out of range, etc..).

The board display is visually clear with great formatting.

Win conditions (horizontal, vertical, diagonal) have repetitive code patterns. These could be generalized instead of rewriting the same logic.

The code is functional and adheres to Connect4 rules effectively.

There is small room for optimization, particularly in handling code repetition.

Numerical Tic-Tac-Toe:

Comprehensive implementation of winning and drawing conditions.

The board display with coordinates is user-friendly and provides clear feedback to players.

Consistent formatting and naming conventions, making the code readable.

However, Overuse of global/static variables (Even, Odd, CurrentSet, IsRandom) violates encapsulation, creating potential for bugs in larger systems.

Random number generation for random indices includes unnecessary loops for generating valid coordinates when the range is already constrained.

This code demonstrates strong foundational design principles and is well-suited for a numerical Tic Tac Toe game. However, global state and redundant code reduce maintainability and scalability. Addressing these issues will make the code more robust and extensible for additional games or features.

4x4 Tic Tac Toe:

The board dimensions and initial state are clearly defined, with a simple 4x4 grid initialization, and tokens ('X' and 'O') are placed in the expected initial positions.

The implementation checks various possible winning conditions (rows, columns, diagonals) using loops, ensuring the game logic covers all bases.

However, the use of global variables like selected, FX, FY, and FourByFour_IsRandom introduces unnecessary side effects that could complicate debugging and testing. It would be better to encapsulate these variables within the relevant classes or game logic.

The win condition checks for rows, columns, and diagonals involve similar code repeated with slight modifications.

The implementation of the 4x4 Tic-Tac-Toe game is functional and logically organized.But, the reliance on global state and redundant code reduces maintainability and clarity. Refactoring to encapsulate state better, simplify win condition checks would improve the code quality.

Loai:

5x5 Tic-Tac-Toe:

display_board method uses formatting to create a visually appealing representation of the board, improving user experience.

The logic for counting win conditions (rows, columns, diagonals) is well-implemented and systematically checks for consecutive matching symbols.

However, the use of global variables such as c, draw, and temp_moves introduces potential side effects and makes the code harder to maintain. These variables should be encapsulated within the class.

The countWin method contains duplicated logic for checking horizontal, vertical, and diagonal wins. This could be refactored into a helper function that checks for consecutive tokens along any line.

The implementation is a solid start for a 5x5 Tic-Tac-Toe game. However, there are several areas for improvement in terms of code structure, clarity, and maintainability. The global state and repetitive logic should be refactored to improve readability and reduce complexity. Making these changes will result in a more robust and easier-to-maintain codebase.

3x3 Tic-Tac-Toe:

The code handles the game rules for the Misère version of Tic-Tac-Toe, which is a variant where the objective is to avoid winning. The turn-based logic is implemented, and the game checks for win conditions correctly.

The display makes the board visually appealing and ensures proper spacing for clarity.

However, the code relies heavily on global variables like turn, lost, won, and moves. These variables could be encapsulated inside a class or as part of the game state to improve maintainability and readability, and the process of winning is split between these variables, which complicates the understanding of game flow.

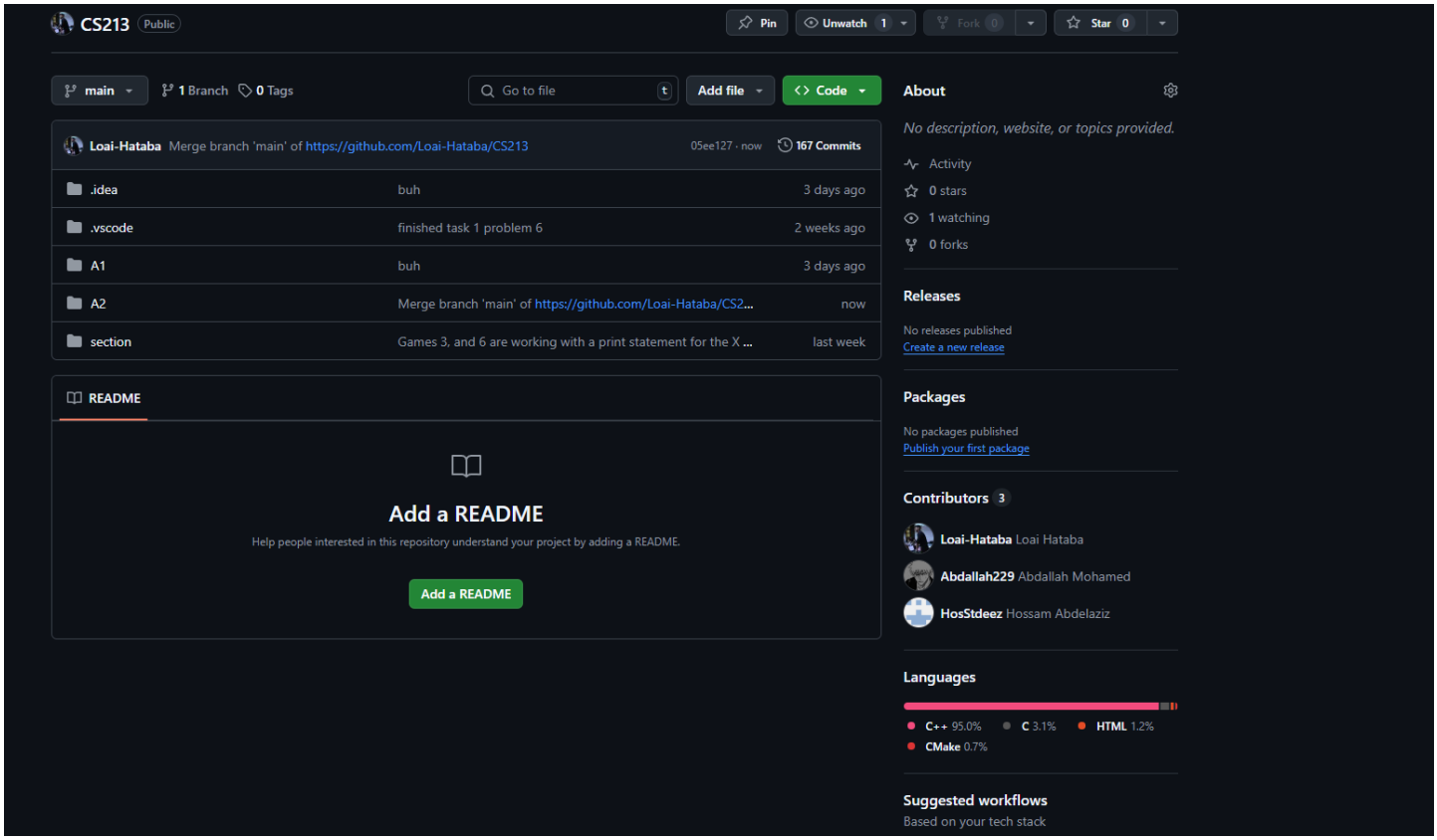
Ultimate Tic-Tac-Toe:

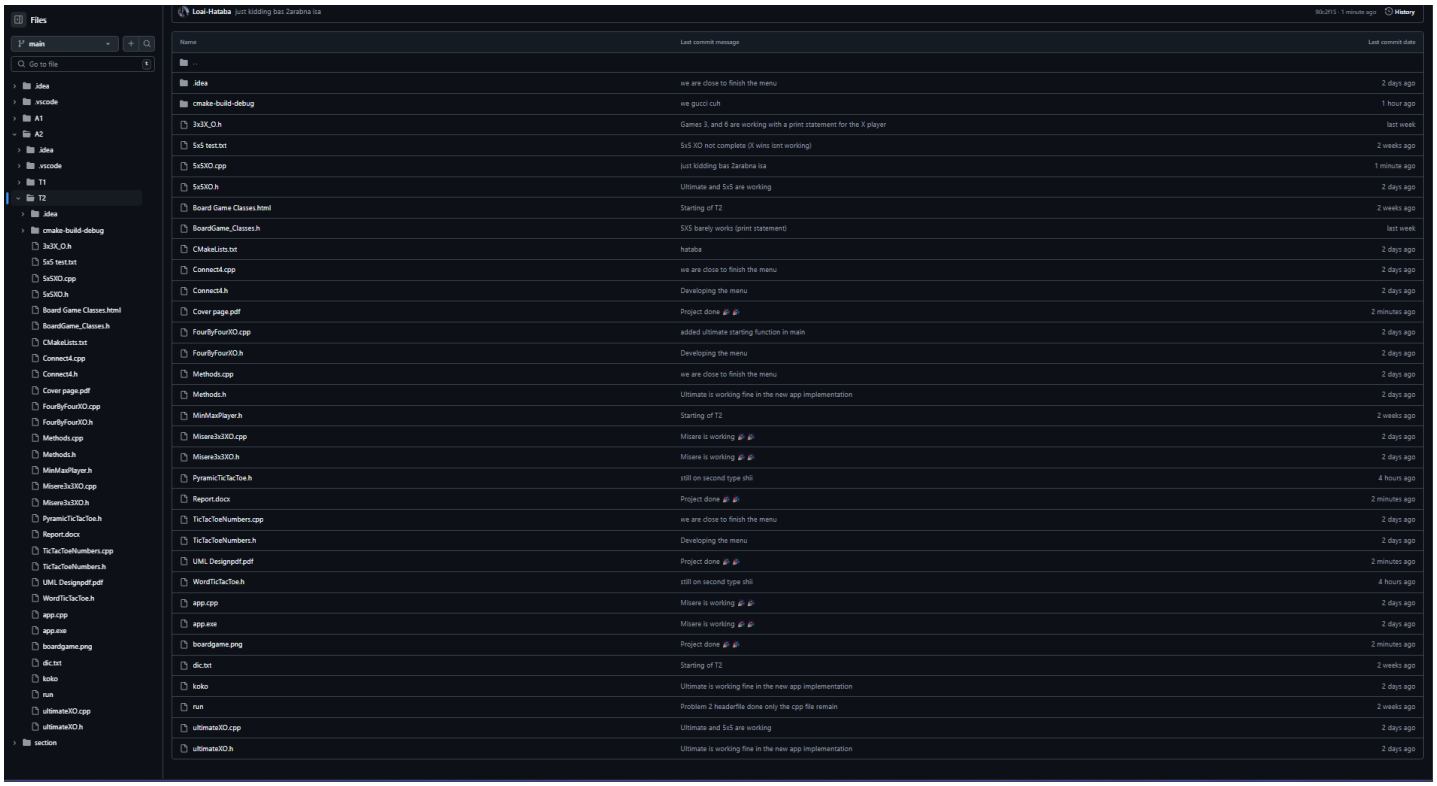
The UltimateXO code looks well-organized for implementing an advanced version of Tic-Tac-Toe, featuring an ultimate board layout with a 9x9 grid split into smaller 3x3 sections.

The display_board() function provides a visually distinct board layout with padding, separators, and alignment, making it easier for players to view the game's state.

However, The RandomPlayer getmove() logic doesn't fully ensure valid moves in sections that aren't already won, leading to potential infinite loops or invalid states during gameplay.

GitHub Repo


CS213
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
Loai-Hataba Merge branch 'main' of https://github.com/Loai-Hataba/CS213
05ee127 · now
167 Commits
.idea buh 3 days ago
.vscode finished task 1 problem 6 2 weeks ago
A1 buh 3 days ago
A2 Merge branch 'main' of https://github.com/Loai-Hataba/CS2... now
section Games 3, and 6 are working with a print statement for the X ... last week
README
Add a README
Help people interested in this repository understand your project by adding a README.
Add a README
About
No description, website, or topics provided.
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 3
Loai-Hataba Loai Hataba
Abdallah229 Abdallah Mohamed
HosStdeez Hossam Abdelaziz
Languages
C++ 95.0%
C 3.1%
HTML 1.2%
CMake 0.7%
Suggested workflows
Based on your tech stack


Files
main
Go to file
.idea
.vscode
A1
A2
.idea
.vscode
T1
T2
.idea
cmake-build-debug
3x3X_O.h
5x5 test.txt
5x5XO.cpp
5x5XO.h
Board Game Classes.html
BoardGame_Classes.h
CMakeLists.txt
Connect4.cpp
Connect4.h
Cover page.pdf
FourByFourXO.cpp
FourByFourXO.h
Methods.cpp
Methods.h
MinMaxPlayer.h
Misere3x3XO.cpp
Misere3x3XO.h
PyramicTicTacToe.h
Report.docx
TicTacToeNumbers.cpp
TicTacToeNumbers.h
UML Designpdf.pdf
WordTicTacToe.h
app.cpp
app.exe
boardgame.png
dic.txt
koko
run
ultimateXO.cpp
ultimateXO.h
section
Loai-Hataba just kidding bas Zarabna isa
Name
Last commit message
Last commit date
.idea we are close to finish the menu 2 days ago
cmake-build-debug we gucci cuh 1 hour ago
3x3X_O.h Games 3, and 6 are working with a print statement for the X player last week
5x5 test.txt 5x5 XO not complete (X wins isnt working) 2 weeks ago
5x5XO.cpp just kidding bas Zarabna isa 1 minute ago
5x5XO.h Ultimate and 5x5 are working 2 days ago
Board Game Classes.html Starting of T2 2 weeks ago
BoardGame_Classes.h 5X5 barely works (print statement) last week
CMakeLists.txt hataba 2 days ago
Connect4.cpp we are close to finish the menu 2 days ago
Connect4.h Developing the menu 2 days ago
Cover page.pdf Project done 2 minutes ago
FourByFourXO.cpp added ultimate starting function in main 2 days ago
FourByFourXO.h Developing the menu 2 days ago
Methods.cpp we are close to finish the menu 2 days ago
Methods.h Ultimate is working fine in the new app implementation 2 days ago
MinMaxPlayer.h Starting of T2 2 weeks ago
Misere3x3XO.cpp Misere is working 2 days ago
Misere3x3XO.h Misere is working 2 days ago
PyramicTicTacToe.h still on second type shii 4 hours ago
Report.docx Project done 2 minutes ago
TicTacToeNumbers.cpp we are close to finish the menu 2 days ago
TicTacToeNumbers.h Developing the menu 2 days ago
UML Designpdf.pdf Project done 2 minutes ago
WordTicTacToe.h still on second type shii 4 hours ago
app.cpp Misere is working 2 days ago
app.exe Misere is working 2 days ago
boardgame.png Project done 2 minutes ago
dic.txt Starting of T2 2 weeks ago
koko Ultimate is working fine in the new app implementation 2 days ago
run Problem 2 headerfile done only the cpp file remain 2 weeks ago
ultimateXO.cpp Ultimate and 5x5 are working 2 days ago
ultimateXO.h Ultimate is working fine in the new app implementation 2 days ago