Faculty of Computers and Artificial Intelligence

Cairo University

CS213: Programming II

Submitted to: Dr. Mohamed El-Ramly

Assignment2_ Task2,3,4,5

Name	ID
Mariam Badr Yahya Abd El-Naby	20230391
Ola Ghoneim Hammad Ahmed	20231232
Menna Mohamed Ashour Ali	20231180

Emails:

mariambadr145@gmail.com

olaghoneim38@gmail.com

mm1881569@gmail.com

Classes Description and UML Class Diagram

Pyramic Tic-Tac-Toe

pyramid_Board Class:

- This class represents a pyramid-shaped game board and inherits from the Board class.

- It initializes the board with a pyramid structure, manages player moves, and checks for win or draw conditions.

- Key methods include update_board, display_board, is_win, is_draw, and game_is_over.

pyramid_player Class:

- This class represents a player in the pyramid game and inherits from the Player class.

- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

pyramid_Random_Player Class:

- This class represents a player that makes random moves in the pyramid game and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's symbol and name, and seeds the random number generator. The getmove method generates random coordinates for moves.

Four_in_row

1. Four_in_row_Board Class:

- This class represents a 6x7 game board for a "Four in a Row" game and inherits from the Board class.

- It initializes the board, manages player moves, and checks for win or draw conditions.

- Key methods include update_board, display_board, is_win, is_draw, and game_is_over.

2. Four_in_row_player Class:

- This class represents a player in the "Four in a Row" game and inherits from the Player class.

- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

3. Four_in_row_Random_Player Class:

- This class represents a player that makes random moves and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's symbol and name, and seeds the random number generator. The getmove method generates random coordinates for moves.

5 x 5 Tic Tac Toe

1. _5x5_Board Class:

- This class represents a 5x5 game board and inherits from the Board class.

- It initializes a 5x5 board and manages the board state, player moves, and game status.

- Key methods include update_board, display_board, is_win, is_draw, game_is_over, little_win, and little_draw.

- The class also tracks scores for players 'X' and 'O' and determines win conditions based on the board state.

2. _5x5_Player Class:

- This class represents a player in the 5x5 board game and inherits from the Player class.

- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

3. _5x5_Random_Player Class:

- This class represents a player that makes random moves and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's name and symbol and seeds the random number generator. The getmove method generates random coordinates for moves.

4- Word Tic-tac-toe

1. word_Board Class

This class represents a 3x3 word-based game board, inheriting from the Board class. It manages the board's state, player moves, and game conditions. Key methods include update_board to place moves, display_board to show the board, is_win to validate words against a dictionary file, is_draw to check for a tie, and game_is_over to determine if the game ends.

2. word_Player Class

This class represents a human player, inheriting from the Player class. It initializes the player's name and symbol and provides the getmove method to input move coordinates.

3. word_Random_Player Class

This class represents a random-move computer player, inheriting from RandomPlayer and word_Board. It initializes the player's symbol and uses the getmove method to generate random move coordinates.

Numerical Tic-Tac-Toe

1. Numerical_Tic_Tac_Toe_board Class:

- This class represents a 3x3 game board for a Numerical Tic-Tac-Toe game and inherits from the Board class.

- It initializes the board, manages player moves, and checks for win or draw conditions.

- Key methods include update_board, display_board, is_win, is_draw, and game_is_over.

2. Numerical_Tic_Tac_Toe_player Class:

- This class represents a player in the Numerical Tic-Tac-Toe game and inherits from the Player class.

- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

3. Numerical_Tic_Tac_Toe_random_player Class:

- This class represents a player that makes random moves in the Numerical Tic-Tac-Toe game and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's symbol and name, and seeds the random number generator. The getmove method generates random coordinates for moves.

Misere Tic Tac Toe

1. Misere_Board Class:

- This class represents a 3x3 game board for a Misere variant of tic-tac-toe and inherits from the Board class.

- It initializes the board, manages player moves, and checks for win or draw conditions.

- Key methods include update_board, display_board, is_win, is_draw, and game_is_over.

2. Misere_Player Class:

- This class represents a player in the Misere tic-tac-toe game and inherits from the Player class.

- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

3. Misere_Random_Player Class:

- This class represents a player that makes random moves in the Misere tic-tac-toe game and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's name and symbol, and seeds the random number generator. The getmove method generates random coordinates for moves.

4 x 4 Tic-Tac-Toe

1. _4x4_Board Class:

- This class represents a 4x4 game board and inherits from the Board class.

- It initializes a 4x4 board with specific symbols ('X' and 'O') in certain positions and provides methods to update the board, display it, and check for win or draw conditions.

- Key methods include update_board, display_board, is_win, is_draw, and game_is_over.

2. _4x4_Player Class:

- This class represents a player in the 4x4 board game and inherits from the Player class.

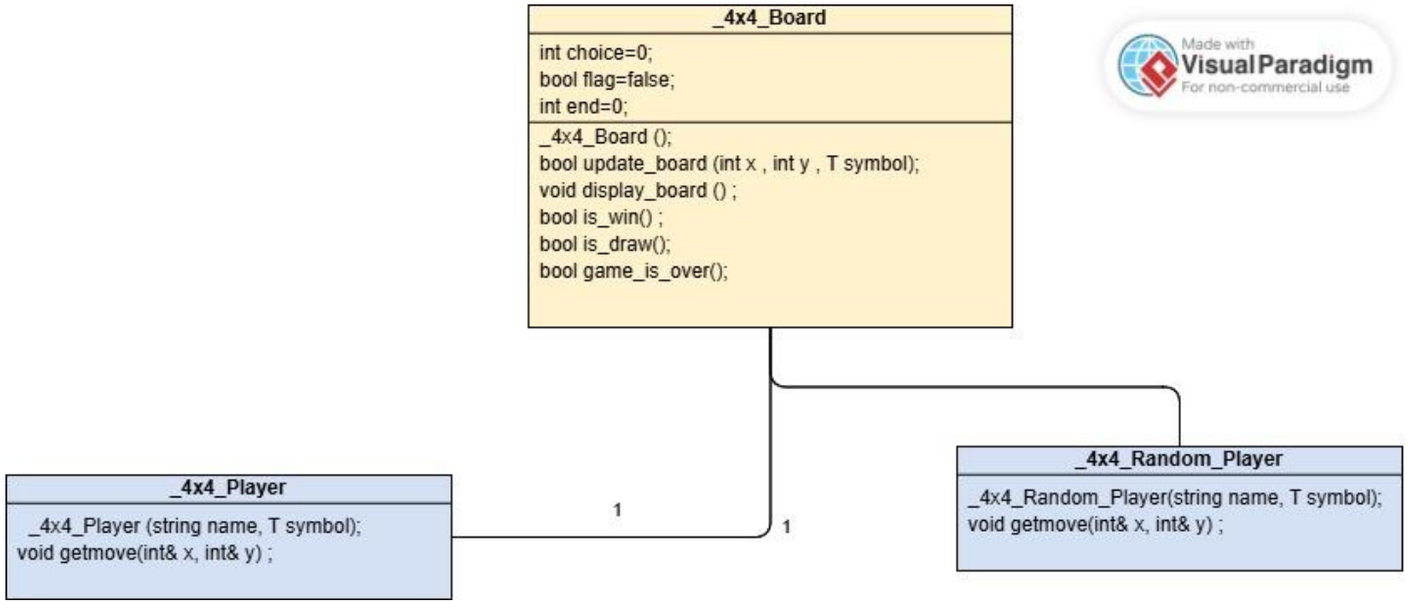
- It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move.

3. _4x4_Random_Player Class:

- This class represents a player that makes random moves and inherits from the RandomPlayer class.

- It includes a constructor to initialize the player's name and symbol and seeds the random number generator. The getmove method generates random coordinates for moves.

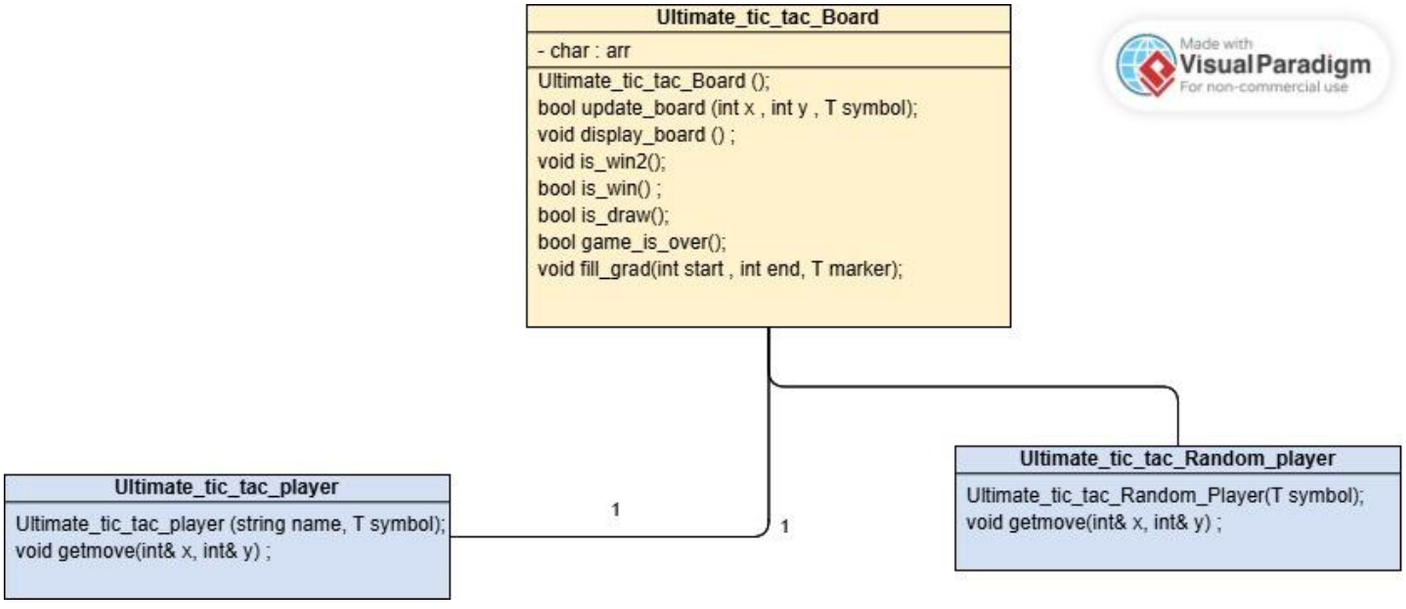
UMl Design:-



Ultimate Tic Tac Toe

1. Ultimate_tic_tac_Board Class:
2.
    - This class represents the game board for Ultimate Tic Tac Toe and inherits from the Board class.
    - It manages the main board with a 9x9 grid, as well as 3 mini-boards for tracking moves.
    - Key methods include update_board, display_board, is_win2, is_win, is_draw, and game_is_over. It handles the game logic, checking for win conditions across rows, columns, diagonals, and mini-boards.
3. Ultimate_tic_tac_player Class:
    - This class represents a player in Ultimate Tic Tac Toe and inherits from the Player class.
    - It includes a constructor to initialize the player's name and symbol, and a method getmove to prompt the player to enter their move on the main board.
4. Ultimate_tic_tac_Random_Player Class:
    - This class represents a player that makes random moves in Ultimate Tic Tac Toe and inherits from the RandomPlayer class.
    - It includes a constructor to initialize the player's symbol and name, seeds the random number generator, and the getmove method generates random coordinates for moves on the main board.

UML Design:-



2. Teamwork Distribution:

A) 20231232:
- Game 3,6,7

B) 20230391:
- Game 1,4
- Menu

C) 20231180:
- Game 2,5,8

***All Team Do Report and Game 9.**

Bonus Part:

What Was Done for the Bonus:

AI for Individual games:

Tools and Libraries Used:

- **Standard C++ Libraries**: The implementation uses <limits> for setting the initial best value for comparisons, and <algorithm> for the std::min and std::max functions to determine the best move.
- **Tic-Tac-Toe Board**: The BoardGame class is utilized to manage the game board, handle the update_board method to simulate moves, and check for win or draw conditions via is_win() and is_draw() methods.

How It Works:

1. **Initialization**: The X_O_MinMax_Player class is initialized with the player's name and symbol (either 'X' or 'O').
2. **Move Decision**: The getmove method calls the getBestMove function to find the optimal move. This move is calculated using the MinMax algorithm, which recursively evaluates the game board's state.
3. **MinMax Calculation**: The calculateMinMax function is called recursively to explore all potential moves. For each move, it simulates the outcome by updating the board and evaluating the opponent's response. The function returns a value based on whether the AI is trying to maximize or minimize its score.
4. **Undoing Moves**: After simulating a move, the update_board method undoes the move to backtrack and explore other options

5. Code Quality Report

	20231180	20231232
Strengths:	**Template Usage:** Both word Board and pyramid Board use templates, allowing for flexibility and extensibility. This shows a strong understanding of generic programming in C++. **Modular Code Structure:** The implementation is well-structured into separate classes for the board, player, random player, and AI player. Reusable methods like update board, display board, and is win are consistently implemented across different board types.	**Game Logic Completeness:** Both board types (word Board and pyramid Board) check for win, draw, and game-over conditions. Comprehensive handling of user input in player classes. **Random Player Implementation:** Random players are implemented with logic to generate moves dynamically, ensuring varied gameplay. **Error Handling:** The word Board::is win method gracefully handles dictionary file errors with appropriate messages.
Issues:	**Memory Management:** word_Board uses raw pointers (new char*) for the board, leading to potential memory leaks. Modern C++ should prefer smart pointers (std::unique_ptr or std::vector). Example: this->board in word_Board should be replaced with a std::vector<std::vector<char>> for automatic memory management. **Random Player Logic for Pyramid Board:**	**Improper Method Call:** In pyramid_Random_Player::getmove, the method this->boardPtr->update_board(x, y, 0) is invoked unnecessarily twice. **File Dependency in word_Board:** The is_win method depends on the external dic.txt file. It lacks a fallback mechanism or pre-validation to handle missing or incorrect file formats.

	Inpyramid_Random_Player::getmove, it redundantly updates the board twice during the same loop iteration..	

Review Process	**Memory Management:** Suggested replacing raw pointers with std::vector. A sample implementation for word_Board was provided. **Random Player Fix:** Modified the logic to ensure update_board is only called once in pyramid_Random_Player::getmove.	**File Handling in word_Board:** Added suggestions to validate file existence and contents at the beginning of the game and provide user feedback. **Display Formatting:** Recommended using formatted output utilities like std::setw to ensure consistent alignment across different screen resolutions

	20231180	20230391
Strengths:	**Game Mechanics Implementation**: The logic for the Tic-Tac-Toe game, including win detection, board updating, and player management, is effectively designed. Methods like `update_board()`, `is_win()`, `is_draw()`, and `game_is_over()` handle core game mechanics well. **Flexible Player Handling**: The design accommodates different types of players (human, random, AI), which adds variety to gameplay.	**Scalable Game Boards**: The game supports both 3x3 and 5x5 boards, making the code more adaptable to different board sizes. **Separation of Concerns**: The use of different classes (`X_O_Board`, `X_O_Player`, `X_O_Random_Player`) to handle various responsibilities (board management, player behavior) follows good object-oriented design principles
Issues:	**Memory Management**: The `X_O_Board` class dynamically allocates memory for the board but lacks proper memory deallocation in some cases (e.g., when using a `delete` for `B` but not ensuring that the allocated memory is freed from within the game class). **Hardcoded Values**: The row and column limits for the 3x3 and 5x5 boards are hardcoded in methods like `update_board()`. This could be refactored for better flexibility and maintainability by using member variables like `this->rows` and `this->columns` instead of repeating `3` or `5` ..	**Functionality Overlap**: There is overlapping functionality between `little_win()`, `is_win()`, and `little_draw()`. This could be simplified or refactored to avoid redundant checks. **Variable Initialization**: In the `X_O_Board` constructor, `players[0] = playerArray[0];` and `players[1] = playerArray[1];` are used, but there is no clear validation or check for the correctness of this assignment. This could lead to issues if the players array isn't properly initialized.

Review Process	**Player Types and Customization**: The ability to choose between different types of players is a great feature, allowing for both human and AI-based gameplay. However, more input validation and error handling would improve user experience and robustness. **Game Over Conditions**: The logic to check if the game is over (`game_is_over()`) is valid, but there could be more clear distinction between conditions where the game ends because of a win versus a draw.	**Code Structure**: The code is well-structured but could benefit from better encapsulation and memory management, especially around dynamic memory allocation and deallocation.

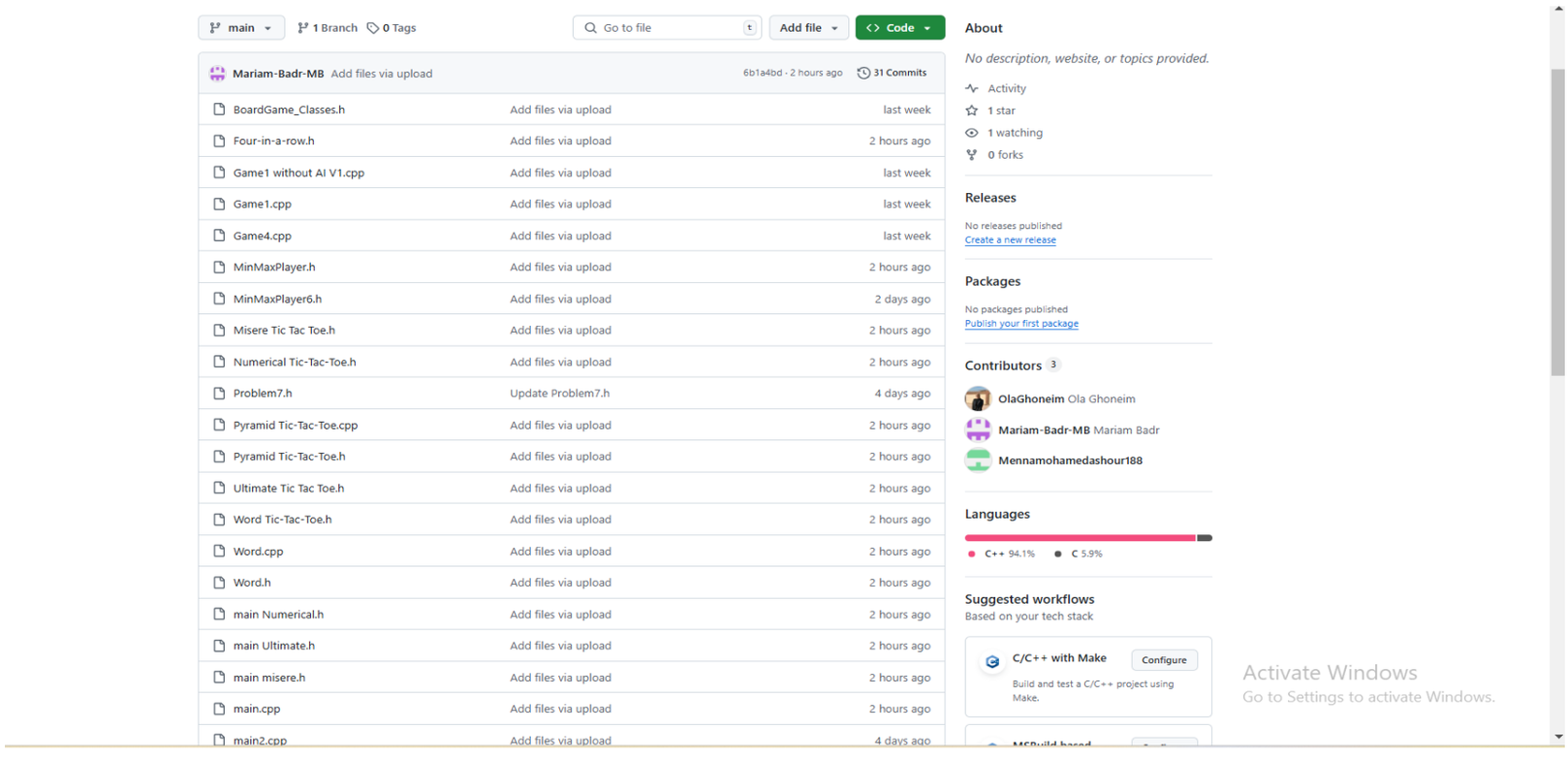
	202321232	20230391
Strengths:	**Object-Oriented Design**: The use of base classes (`Board`, `Player`, `RandomPlayer`) and derived classes (`NumericalTic_Tac_Toe_board`, `Four_in_row_Board`, etc.) adheres to the principles of inheritance and polymorphism. **Dynamic Memory Management**: Dynamic memory allocation for game boards is handled using `new` and pointers, providing flexibility for varying game board sizes	**Input Validation**: Functions such as `update_board` include checks for invalid inputs, ensuring robustness against incorrect user or system inputs. **Readable Structure**: The code is modular, with separate classes and functions for players, game boards, and game management. **User Interaction**: Prompts and feedback to the user enhance usability (e.g., "Invalid column!" or "Please enter x and y").
Issues:	**Memory Management Risks**: `delete` is used to free allocated memory, but there is no exception handling to prevent memory leaks during unexpected terminations. Inconsistent cleanup: Objects like players are deleted, but there is no explicit destructor for classes managing dynamically allocated arrays. **Code Duplication**: The player setup logic for both games is repeated in `main`. This could be extracted into a reusable function to reduce redundancy. ..	**Unnecessary Attributes**:Attributes like `name` in player classes are set redundantly in constructors. **Error Handling**: Lack of error handling for invalid user inputs (e.g., non-integer input for `choice` or moves). **Overhead in `RandomPlayer`**: The `srand` function is called in the constructor, potentially leading to issues if multiple `RandomPlayer` instances are created quickly.

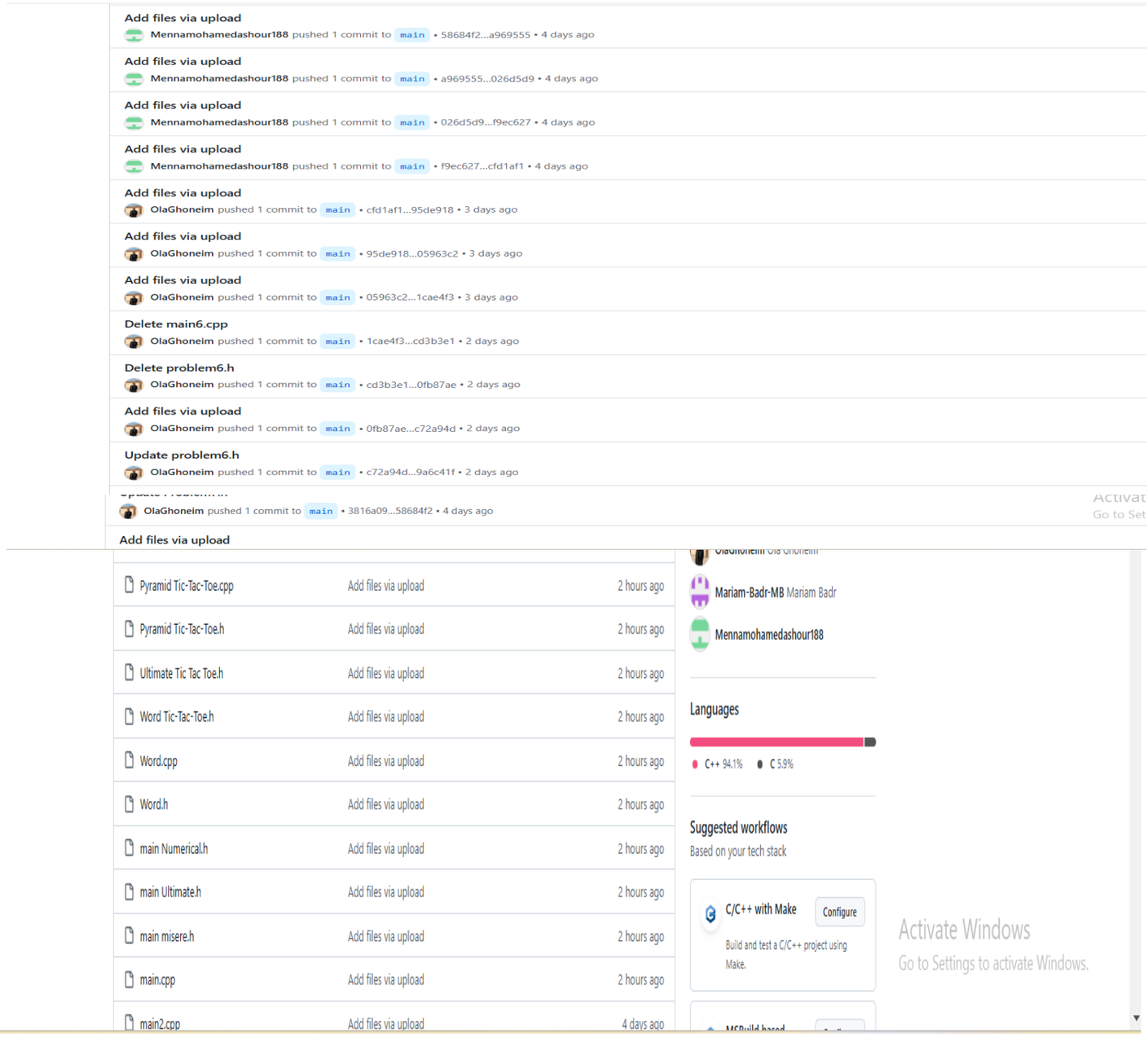
Review Process	Player Types and **The** issues were identified by tracing the flow of the program and testing edge cases such as invalid inputs, memory deallocation, and gameplay scenarios. Key inefficiencies, like code duplication, were observed by comparing similar logic across functions and classes	Recommendations were discussed and resolved by proposing modular functions, exception handling, and adherence to DRY principles. .

GitHub Activity:

Repo Link:

https://github.com/Mariam-Badr-MB/8-games




Add files via upload
Mennamohamedashour188 pushed 1 commit to main • 58684f2...a969555 • 4 days ago
Add files via upload
Mennamohamedashour188 pushed 1 commit to main • a969555...026d5d9 • 4 days ago
Add files via upload
Mennamohamedashour188 pushed 1 commit to main • 026d5d9...f9ec627 • 4 days ago
Add files via upload
Mennamohamedashour188 pushed 1 commit to main • f9ec627...cfd1af1 • 4 days ago
Add files via upload
OlaGhoneim pushed 1 commit to main • cfd1af1...95de918 • 3 days ago
Add files via upload
OlaGhoneim pushed 1 commit to main • 95de918...05963c2 • 3 days ago
Add files via upload
OlaGhoneim pushed 1 commit to main • 05963c2...1cae4f3 • 3 days ago
Delete main6.cpp
OlaGhoneim pushed 1 commit to main • 1cae4f3...cd3b3e1 • 2 days ago
Delete problem6.h
OlaGhoneim pushed 1 commit to main • cd3b3e1...0fb87ae • 2 days ago
Add files via upload
OlaGhoneim pushed 1 commit to main • 0fb87ae...c72a94d • 2 days ago
Update problem6.h
OlaGhoneim pushed 1 commit to main • c72a94d...9a6c41f • 2 days ago
OlaGhoneim pushed 1 commit to main • 3816a09...58684f2 • 4 days ago
Add files via upload
Pyramid Tic-Tac-Toe.cpp Add files via upload 2 hours ago
Pyramid Tic-Tac-Toe.h Add files via upload 2 hours ago
Ultimate Tic Tac Toe.h Add files via upload 2 hours ago
Word Tic-Tac-Toe.h Add files via upload 2 hours ago
Word.cpp Add files via upload 2 hours ago
Word.h Add files via upload 2 hours ago
main Numerical.h Add files via upload 2 hours ago
main Ultimate.h Add files via upload 2 hours ago
main misere.h Add files via upload 2 hours ago
main.cpp Add files via upload 2 hours ago
main2.cpp Add files via upload 4 days ago
Mariam-Badr-MB Mariam Badr
Mennamohamedashour188
Languages
C++ 94.1%
C 5.9%
Suggested workflows
Based on your tech stack
C/C++ with Make
Configure
Build and test a C/C++ project using Make.
Activate Windows
Go to Settings to activate Windows.