CS213 - Object Oriented Programming

Dr: Mohamed Elramly

Assignment 2 – task 2 & 3

Name:	ID:
Kareem Ahmed	20230293
Ahmed Saeed	20230020
Ahmed Khalil	20230017

1- **the Pyramid Tic-Tac-Toe** game and how it works, broken down into easy-to-understand steps:

How It Works:

1. **The Board**:

   This game is played on a **pyramid-shaped board** with 3 rows:

   The 1st row has 1 cell.

   The 2nd row has 3 cells.

   The 3rd row has 5 cells.

   Each cell starts empty, and players take turns placing their symbols (X or O) on the board.

2. **Players**:

   Two players can participate:

   Human Player: Manually selects moves.

   Random Computer Player: Automatically picks random valid moves.

3. **Gameplay**:

   Players alternate turns to place their symbol in an empty cell.

   A move is valid if it is within the bounds of the pyramid and the chosen cell is empty.

   After each turn, the updated pyramid board is displayed.

4. **Win Condition**:

   A player wins if they form a streak of three consecutive symbols (X or O) in:

   A horizontal row.

   A diagonal (slanting from one row to the next).

        - A vertical column.

    - The program checks all possible winning patterns in the pyramid.

5. **Draw Condition**:

    - If all 9 cells in the pyramid are filled and no one has formed a winning streak, the game ends in a draw.

6. **Game Manager**:

    - The main program handles the setup of the board and players.
    - It manages player turns, checks for wins or draws, and declares the outcome (win or draw) once the game ends.

What Makes It Unique:

- Unlike a regular Tic-Tac-Toe board, this game uses a **pyramid** structure, adding complexity to the gameplay.
- The board's layout changes row-by-row, which means players must think about positioning their symbols differently compared to a flat grid.
- The randomness of the computer player introduces unpredictability if one or both players are automated.

Example Gameplay:

1. **Setup**:

    - Player 1 chooses to play as a Human with symbol X.
    - Player 2 chooses the Random Computer with symbol O.

2. **Turn-by-Turn**:

    - Player 1 places an X in the middle of the second row.
    - Player 2 (Random Computer) selects a random cell and places an O.
    - The game continues until a win or draw condition is met.

3. **Game Over**:

    If Player 1 forms a horizontal streak of three X symbols in the 3rd row, they win.

    If no winning streak is formed and the board is full, the game ends in a draw.

Here’s a concise explanation of how the **Four-in-a- Row** game works, along with a simple breakdown of its components:

How It Works

1. **Objective**:
   Players take turns dropping their pieces ('X' or 'O') into one of the seven columns.

    The goal is to connect **four pieces in a row** (horizontally, vertically, or diagonally).

2. **Game Flow**:

    Players alternate turns until:

        Someone wins by connecting four pieces.

        The board fills up, resulting in a draw.

   At the end of each move:

        The board is displayed.

        The program checks for a winner or draw.

Components Explained

1. **Game Board (X_O_Board)**:

    A **6x7 grid** initialized as empty (all cells set to 0).

    Handles:

Placing a piece in the correct column.

Displaying the board to the players.

Checking for a win or draw.

2. **Players**:

    Human Player (X_O_Player):
    Prompts the user to enter their move (row and column).

    Random Player (X_O_Random_Player):
    Simulates a computer player that chooses moves randomly.

3. **Game Rules**:

    Valid Move:

    Players can only drop pieces into columns that are not full.

    Winning Move:

    Four pieces in a line (horizontal, vertical, or diagonal).

    Draw:

    The game ends in a draw if all 42 cells are filled without a winner.

4. **Game Manager (GameManager)**:

    Controls the main game loop:

    Alternates between players.

    Updates the board.

    Ends the game when there’s a winner or draw.

How to Play

1. **Choose Player Types**:

    Human or Random Computer for both players.

2. **Gameplay**:

Human players enter their move as row and column numbers.

Computer players make random moves.

3. **Win or Draw**:

   The game announces the result (winner or draw) at the end.

3-This program is a **5x5 Tic Tac Toe game**, designed for two players to compete, either as humans or against a random computer player.

Here's how it works:

How It Works:

1. **The Board**:

   The game uses a 5x5 grid.

   Each cell is initially empty, and players take turns placing their symbol (X or O) on the board.

2. **Players**:

   Two players participate, and you can choose the type for each:

   Human Player: Manually enters moves.

   Random Computer Player: Picks random positions on the board.

3. **Gameplay**:

   Players take turns to make a move.

   A move is valid only if it's within the grid and the chosen cell is empty.

   After every move, the board updates and displays the current state.

4. **Win and Draw Conditions**:

    Unlike traditional Tic Tac Toe, there's no single winner immediately after forming a streak.

    The game counts how many **three-in-a-row streaks** (X or O) are formed across the board (horizontally, vertically, or diagonally).

    The player with the **most streaks** wins.

    If the board fills up without either player forming enough streaks to outscore the other, it's a draw.

5. **Game Manager**:

    The main program sets up the game, handles player turns, and displays the result:

    It declares the winner, or a draw based on the number of three-in-a-row streaks.

Simple Explanation:

This is a fun twist on regular Tic Tac Toe:

It's played on a larger 5x5 grid.

Instead of one winner, it counts streaks of three symbols (X or O).

You can play as a human or let the computer randomly choose its moves.

At the end of the game, the player with the most streaks wins, or it's a draw if the board fills up evenly.

4-Here's a simplified explanation of how the **Word Board Game** works, broken down into its core components and functionality:

How It Works

1. **Objective**:

   The goal of the game is for players to place letters on a **3x3** grid (like a Tic-Tac-Toe board).

   Players take turns to place letters ('X' or 'O') into the grid.

   To win, a player must form a valid word (from a dictionary) either horizontally, vertically, or diagonally.

2. **Game Flow**:

   Player Setup:
   Players can either choose to play as a **human** or as a **random computer player**.

   Turn-Based Gameplay:

   Player’s alternate placing a letter on the board at a specific row and column.

   After each move, the board is displayed, and the game checks for a winner or a draw.

   The game ends when:

   A player forms a valid word from the dictionary (winning the game).

   The grid is full (resulting in a draw).

Components Explained

1. **Word Board (Word_Board)**:

   A **3x3 grid** is initialized, each cell starts as empty (set to 0).

   It can:

   Update a cell with a letter.

   Display the board.

   Check for a winning word (from a dictionary).

   Check if the game is a draw.

2. **Players**:

    Human Player (Word_Player):
    Prompts the user for their move and the letter they want to place.

    Random Player (Word_RandomPlayer):
    A computer-controlled player that selects a random move and letter.

3. **Winning Condition**:

    The game checks if any row, column, or diagonal forms a valid word from the dictionary (dic.txt).

    If found, that player wins.

4. **Game Manager (GameManager)**:

    Controls the flow of the game:

        Alternates between players.

        Updates the board with each move.

        Ends the game if there's a winner or if the board is full.

How to Play

1. **Choose Player Types**:

    Human or Random Computer for both players.

2. **Gameplay**:

    Players take turns entering their move:

        The player selects a row (0 to 2) and column (0 to 2) to place their letter.

        A letter is also chosen, and the game checks if the move is valid.

3. **Win or Draw**:

    The game checks for a valid word from the dictionary after each move.

The game announces the winner if a valid word is formed or declares a draw if the board is filled with no winner.

Key Features

File-based Dictionary:
The game reads words from a file (dic.txt), and the game will check for valid words based on this list.

Random Player Logic:
The computer-controlled player randomly selects both the position and the letter for its move.

5-This is a **Numerical Tic-Tac-Toe Game** where players can place numbers on a 3x3 grid, and the objective is to achieve a specific sum (15) across a row, column, or diagonal. Here's an explanation of how the code works:

Main Features and Components

1. **Numerical_Board**:

   This class represents the 3x3 game board, which is initialized with zeros.

   Each player can place a number (either 1, 3, 5, 7, or 9 for Player 1, and 2, 4, 6, or 8 for Player 2) on the board.

   The **update_board()** function places a number on the board at the specified coordinates, ensuring no repetition of numbers on the board and the validity of the move.

   The **is_win()** function checks if any row, column, or diagonal sums up to 15. This is the winning condition.

   is_draw() : checks if all moves are made and no one has won.

   game_is_over() returns whether the game has ended (either by win or draw).

2. **Numerical Player**:

This class represents a human player. The player enters their move (coordinates and the number to place) interactively through the console.

The **getmove()** function asks for the player's move and ensures that the chosen number is valid (within the allowed set and not repeated on the board).

3. **Numerical Rand Player**:

    This class represents a computer player. It selects a random position on the board and randomly chooses a valid number to place.

    The **getmove()** function generates random coordinates and ensures that the number follows the same rules as the human player.

How the Game Works

1. **Setup**:

    The game asks for names and types of players (either human or computer) for both players (Player 1 and Player 2).

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

2. **Gameplay**:

    Players take turns placing numbers on the board at specified coordinates. The board is updated after each move.

    The board is displayed after each move to show the current state of the game.

    The game checks after each move whether there is a winner (a row, column, or diagonal sums to 15) or if the game is a draw (board is full with no winner).

3. **Winning Condition**:

    The first player to get a sum of 15 in a row, column, or diagonal wins the game.

The game ends when a player wins or all the cells are filled without a winner (resulting in a draw).

4. **Random Player Logic**:

   For the random computer player, the game generates a random position and ensures the number is valid for that player (odd/even rules).

Game Management:

The Game Manager class controls the game flow, managing turns, checking for game status (win, draw), and displaying the board.

Summary:

Numerical Board: Handles the game board and win condition.

Numerical Player: Manages human player input.

Numerical Rand Player: Handles random computer moves.

Main Function: Initializes the players and the game board, starts the game, and handles cleanup.

Key Notes

The game uses numbers instead of 'X' and 'O'.

The winning condition is a sum of 15 for a row, column, or diagonal.

players can choose to play either as humans or as random computer players.

6- **Misere Tic-Tac-Toe Game**, where the goal is to *avoid forming a line* of matching symbols (row, column, or diagonal). Here's a summary:

Key Features:

1. **Misere Board**:

    Manages the 3x3 board, updates move, and checks for game over conditions.

    is_win() determines if any line (row, column, diagonal) has matching symbols.

    is_draw() handles the case where the board is full without a win.

2. **Player Classes**:

    Misere player: Takes user input for moves.

    Misere random player: Generates random moves for the computer.

3. **Main Function**:

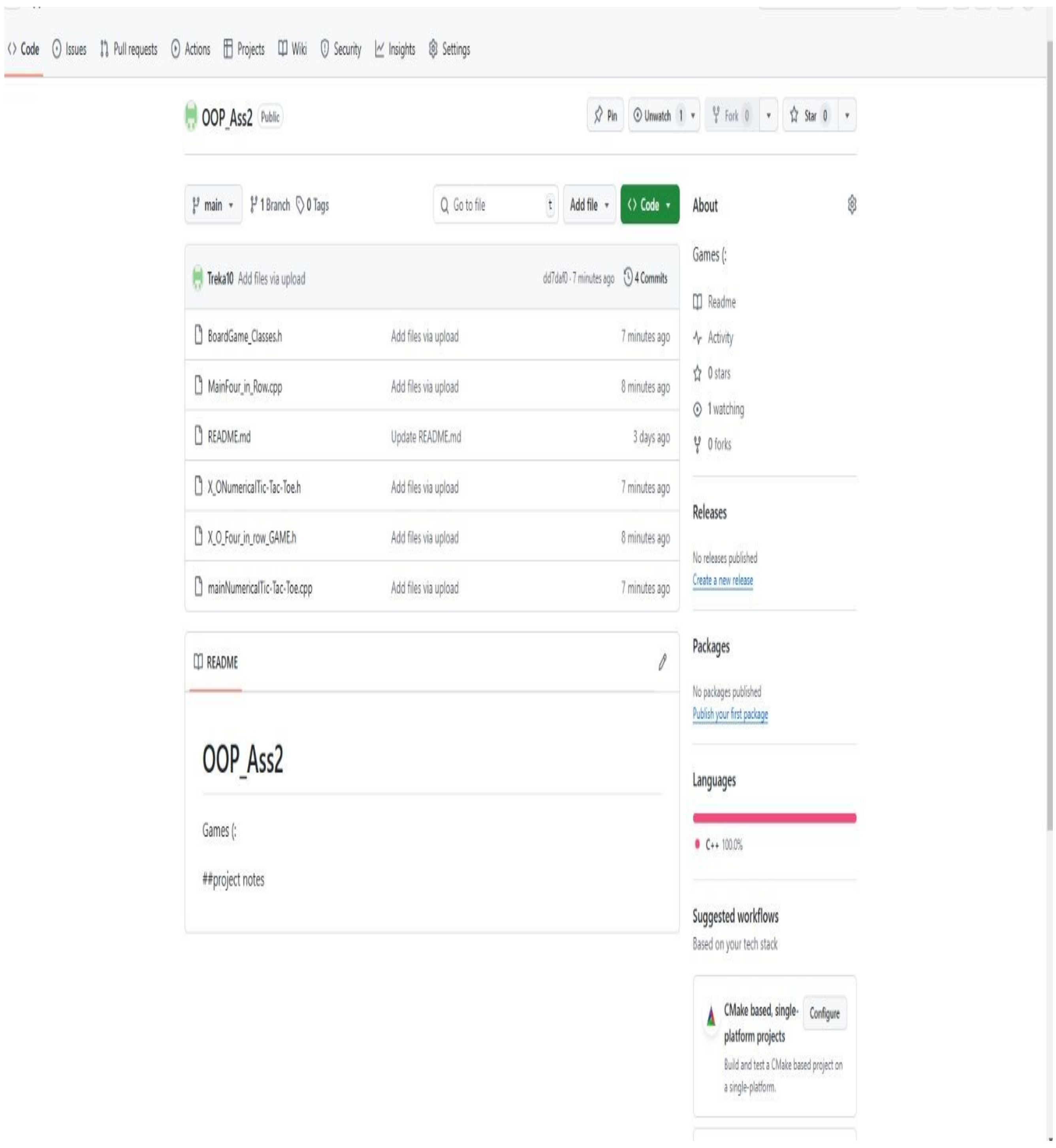
    Allow configuring two players (Human or Random Computer).

    Uses **Game Manager** to handle gameplay, alternating turns, and checking win/draw conditions.

Notes:

The **Misere condition** (losing by completing a line) isn't enforced yet. You can modify **is_win()** logic to check for losing conditions explicitly.

Input validation is minimal; ensure moves are within bounds and avoid overwriting symbols.



OOP_Ass2 Public

Treka10 Add files via upload dd7daf0 · 7 minutes ago 4 Commits

BoardGame_Classes.h	Add files via upload	7 minutes ago
MainFour_in_Row.cpp	Add files via upload	8 minutes ago
README.md	Update README.md	3 days ago
X_ONumericalTic-Tac-Toe.h	Add files via upload	7 minutes ago
X_O_Four_in_row_GAME.h	Add files via upload	8 minutes ago
mainNumericalTic-Tac-Toe.cpp	Add files via upload	7 minutes ago

README

OOP_Ass2

Games (:

##project notes

About

Games (:

Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

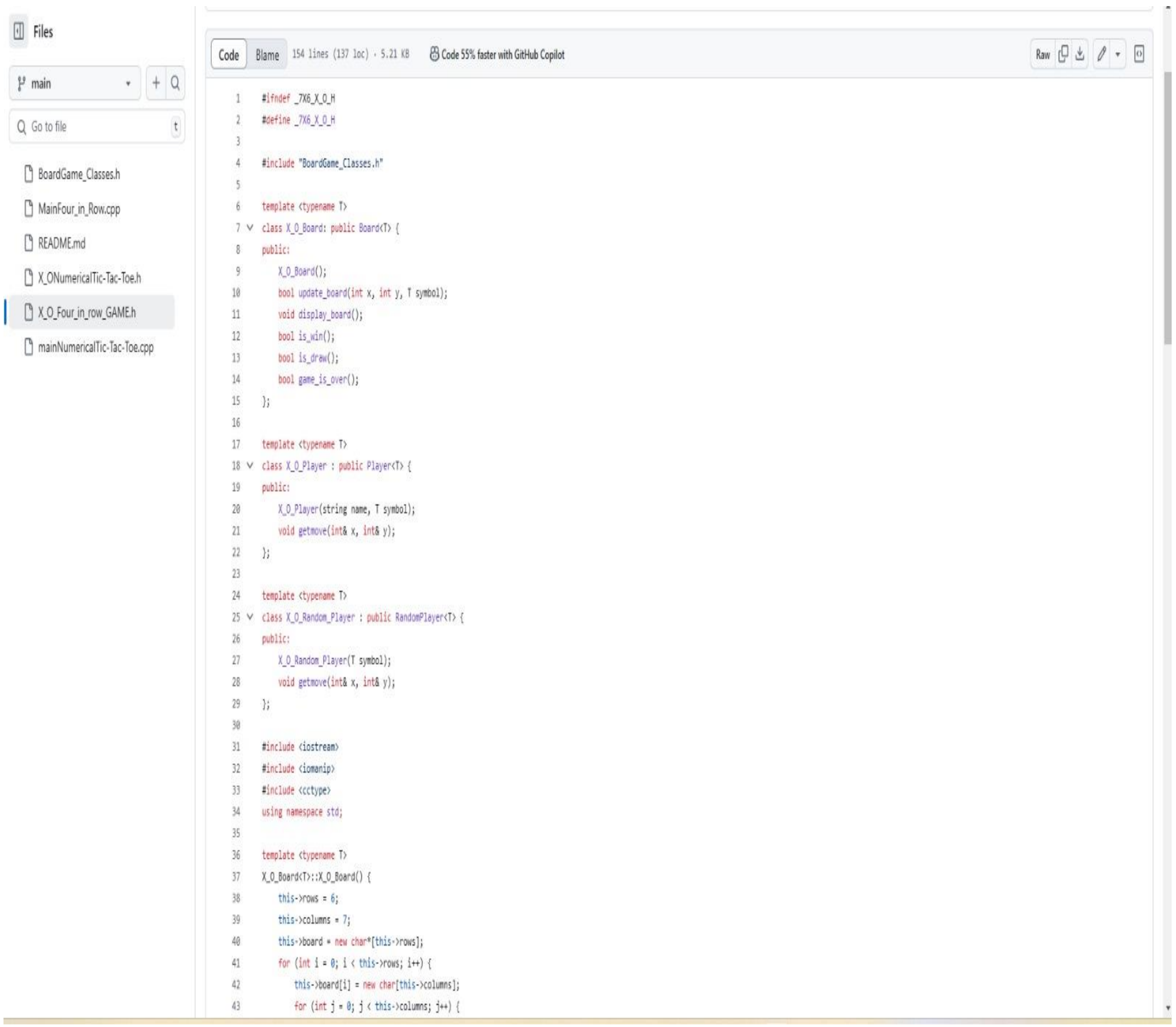
● C++ 100.0%

Suggested workflows

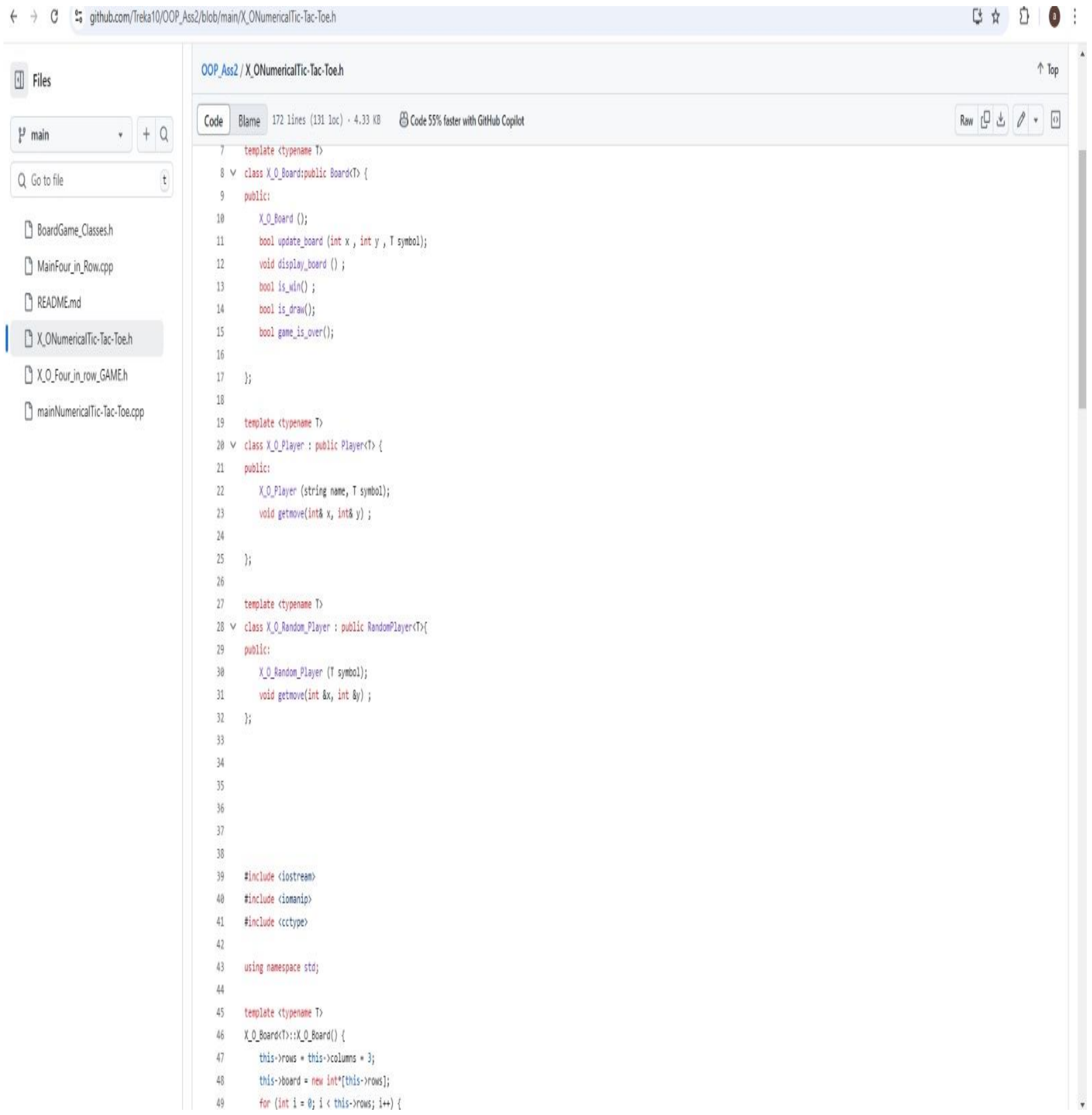
Based on your tech stack

CMake based, single-platform projects — Configure

Build and test a CMake based project on a single-platform.

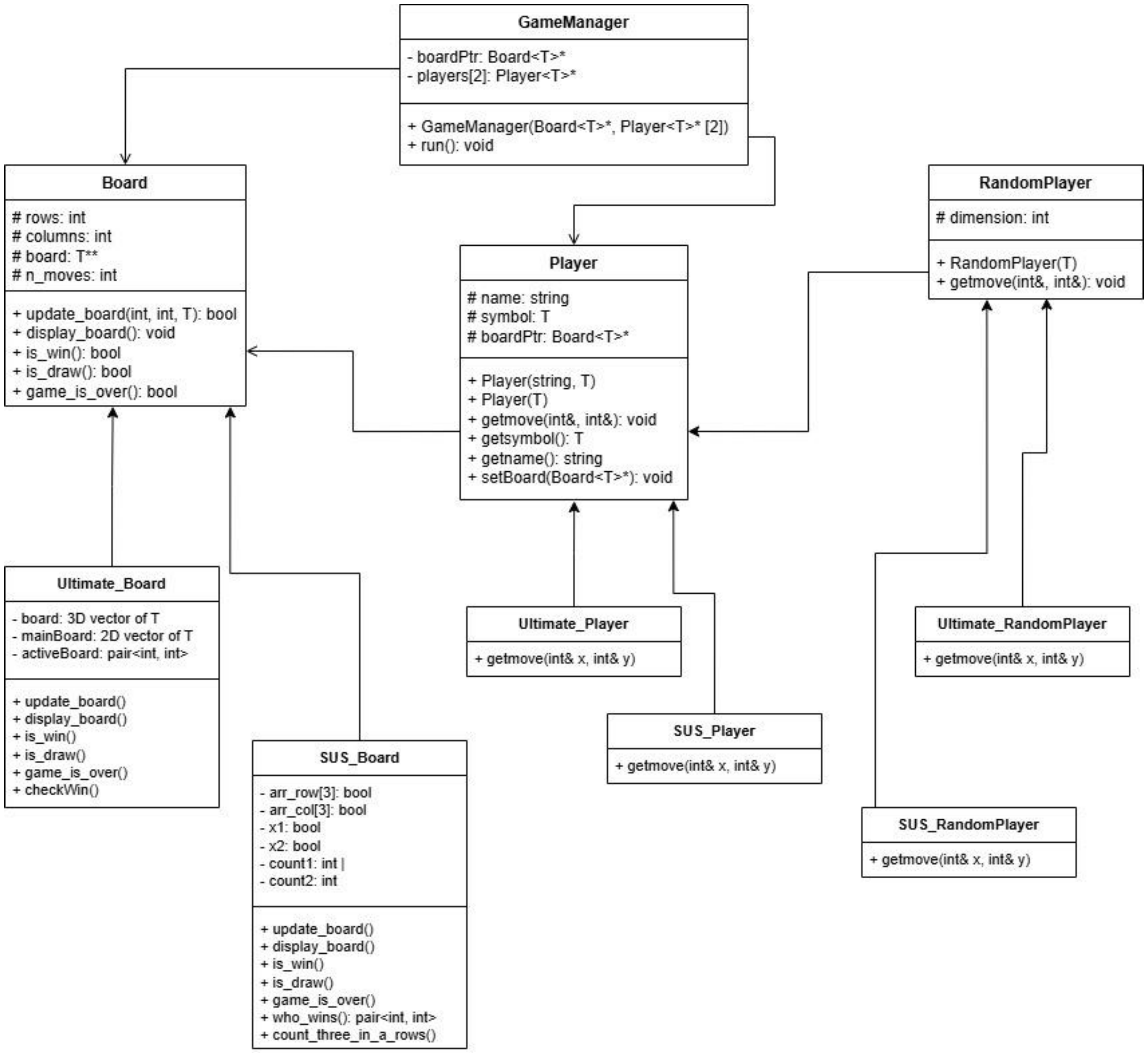

Files
main
Go to file
BoardGame_Classes.h
MainFour_in_Row.cpp
README.md
X_ONumericalTic-Tac-Toe.h
X_O_Four_in_row_GAME.h
mainNumericalTic-Tac-Toe.cpp
Code
Blame
154 lines (137 loc) · 5.21 KB
Code 55% faster with GitHub Copilot
Raw

```cpp
#ifndef _7X6_X_O_H
#define _7X6_X_O_H

#include "BoardGame_Classes.h"

template <typename T>
class X_O_Board: public Board<T> {
public:
    X_O_Board();
    bool update_board(int x, int y, T symbol);
    void display_board();
    bool is_win();
    bool is_draw();
    bool game_is_over();
};

template <typename T>
class X_O_Player : public Player<T> {
public:
    X_O_Player(string name, T symbol);
    void getmove(int& x, int& y);
};

template <typename T>
class X_O_Random_Player : public RandomPlayer<T> {
public:
    X_O_Random_Player(T symbol);
    void getmove(int& x, int& y);
};

#include <iostream>
#include <iomanip>
#include <cctype>
using namespace std;

template <typename T>
X_O_Board<T>::X_O_Board() {
    this->rows = 6;
    this->columns = 7;
    this->board = new char*[this->rows];
    for (int i = 0; i < this->rows; i++) {
        this->board[i] = new char[this->columns];
        for (int j = 0; j < this->columns; j++) {
```


github.com/Treka10/OOP_Ass2/blob/main/X_ONumericalTic-Tac-Toe.h
Files
main
Go to file
BoardGame_Classes.h
MainFour_in_Row.cpp
README.md
X_ONumericalTic-Tac-Toe.h
X_O_Four_in_row_GAME.h
mainNumericalTic-Tac-Toe.cpp
OOP_Ass2 / X_ONumericalTic-Tac-Toe.h
Top
Code
Blame
172 lines (131 loc) · 4.33 KB
Code 55% faster with GitHub Copilot
Raw

```cpp
template <typename T>
class X_O_Board:public Board<T> {
public:
    X_O_Board ();
    bool update_board (int x , int y , T symbol);
    void display_board () ;
    bool is_win() ;
    bool is_draw();
    bool game_is_over();

};

template <typename T>
class X_O_Player : public Player<T> {
public:
    X_O_Player (string name, T symbol);
    void getmove(int& x, int& y) ;

};

template <typename T>
class X_O_Random_Player : public RandomPlayer<T>{
public:
    X_O_Random_Player (T symbol);
    void getmove(int &x, int &y) ;
};






#include <iostream>
#include <iomanip>
#include <cctype>

using namespace std;

template <typename T>
X_O_Board<T>::X_O_Board() {
    this->rows = this->columns = 3;
    this->board = new int*[this->rows];
    for (int i = 0; i < this->rows; i++) {
```

7-this is the class diagram of **sus game and ultimate game**:



who did what:

Kareem ahmed worked on problems 3 & 6 & 9 and the report.

Ahmed Saeed worked on problems 2 & 5 & 8 and the GitHub.

Ahmed Khalil worked on problems 1 & 4 & 8 & 9 & main.