FACULTY OF COMPUTERSCIENCE AND ARTIFICIAL INTELLIGENCE – CAIRO UNIVERSITY

Assignment 2- Tasks 2,3,4,5 Report

DECEMBER 16, 2024

20230593 – 20230587 - 20230591

Abdelrahman Ahmed Samir – Mazen Mohamed Abdelsalam – Yousef Osama Mamdouh

Table of Contents

Classes Description

GameName_Player (Derived from Player)

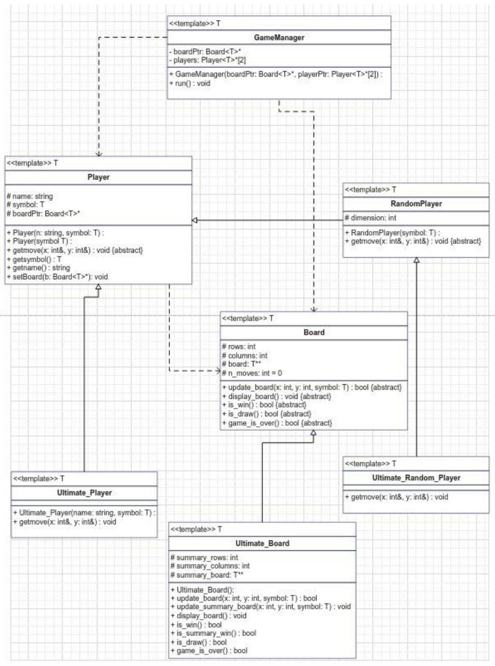
Overrides the getmove method to determine moves as each game has its own special format like including a certain character or number instead of X or O or larger board like 6x7 in Four in a row or 9x9 in Ultimate Tic Tac toe.

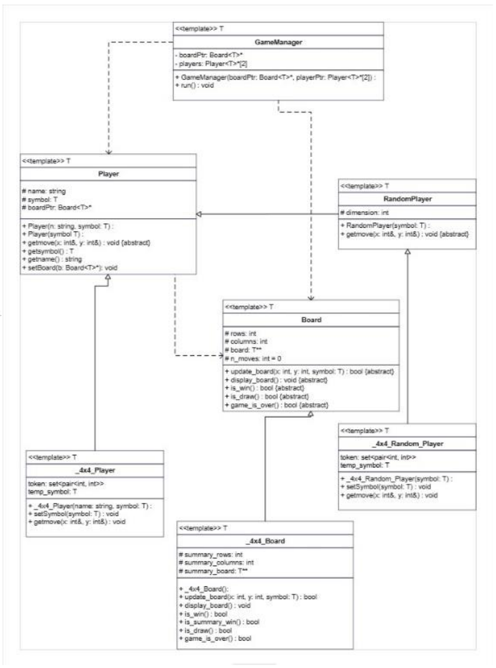
GameName_Random_Player (Derived from RandomPlayer)

Implements random move generation specifically tailored for the game with same constraints as the normal player.

GameName_Board (Derived from Board)

- Updates the board as the game rules state.
- Checks game-over conditions for the entire game.
- Adds logic to manage the rules and structure of **Ultimate Tic-Tac-Toe**:
  - Tracks multiple sub-boards (summary_rows, summary_columns).
  - Implements update_summary_board for handling moves across boards.


<<template>> T
GameManager
- boardPtr: Board<T>*
- players: Player<T>*[2]
+ GameManager(boardPtr: Board<T>*, playerPtr: Player<T>*[2]) :
+ run() : void
<<template>> T
Player
# name: string
# symbol: T
# boardPtr: Board<T>*
+ Player(n: string, symbol: T) :
+ Player(symbol T) :
+ getmove(x: int&, y: int&) : void {abstract}
+ getsymbol() : T
+ getname() : string
+ setBoard(b: Board<T>*): void
<<template>> T
RandomPlayer
# dimension: int
+ RandomPlayer(symbol: T) :
+ getmove(x: int&, y: int&) : void {abstract}
<<template>> T
Board
# rows: int
# columns: int
# board: T**
# n_moves: int = 0
+ update_board(x: int, y: int, symbol: T) : bool {abstract}
+ display_board() : void {abstract}
+ is_win() : bool {abstract}
+ is_draw() : bool {abstract}
+ game_is_over() : bool {abstract}
<<template>> T
Ultimate_Player
+ Ultimate_Player(name: string, symbol: T) :
+ getmove(x: int&, y: int&) : void
<<template>> T
Ultimate_Random_Player
+ getmove(x: int&, y: int&) : void
<<template>> T
Ultimate_Board
# summary_rows: int
# summary_columns: int
# summary_board: T**
+ Ultimate_Board():
+ update_board(x: int, y: int, symbol: T) : bool
+ update_summary_board(x: int, y: int, symbol: T) : void
+ display_board() : void
+ is_win() : bool
+ is_summary_win() : bool
+ is_draw() : bool
+ game_is_over() : bool


<<template>> T
GameManager
- boardPtr: Board<T>*
- players: Player<T>*[2]
+ GameManager(boardPtr: Board<T>*, playerPtr: Player<T>*[2]) :
+ run() : void
<<template>> T
Player
# name: string
# symbol: T
# boardPtr: Board<T>*
+ Player(n: string, symbol: T) :
+ Player(symbol T) :
+ getmove(x: int&, y: int&) : void {abstract}
+ getsymbol() : T
+ getname() : string
+ setBoard(b: Board<T>*): void
<<template>> T
RandomPlayer
# dimension: int
+ RandomPlayer(symbol: T) :
+ getmove(x: int&, y: int&) : void {abstract}
<<template>> T
Board
# rows: int
# columns: int
# board: T**
# n_moves: int = 0
+ update_board(x: int, y: int, symbol: T) : bool {abstract}
+ display_board() : void {abstract}
+ is_win() : bool {abstract}
+ is_draw() : bool {abstract}
+ game_is_over() : bool {abstract}
<<template>> T
_4x4_Player
token: set<pair<int, int>>
temp_symbol: T
+ _4x4_Player(name: string, symbol: T) :
+ setSymbol(symbol: T) : void
+ getmove(x: int&, y: int&) : void
<<template>> T
_4x4_Random_Player
token: set<pair<int, int>>
temp_symbol: T
+ _4x4_Random_Player(symbol: T) :
+ setSymbol(symbol: T) : void
+ getmove(x: int&, y: int&) : void
<<template>> T
_4x4_Board
# summary_rows: int
# summary_columns: int
# summary_board: T**
+ _4x4_Board():
+ update_board(x: int, y: int, symbol: T) : bool
+ display_board() : void
+ is_win() : bool
+ is_summary_win() : bool
+ is_draw() : bool
+ game_is_over() : bool

Work Breakdown

Abdelrahman

- Games 3, 6
- The Menu
- The GUI

Mazen

- Games 1, 4, 8
- Class Diagrams

Yousef

- Games 2, 5, 7
- This report

Code Reviews

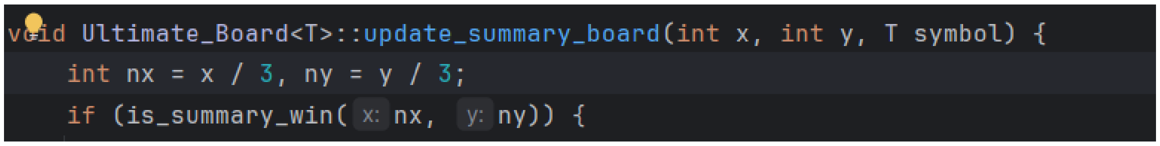
Abdelrahman

Pyramic, Word, and Ultimate Tic Tac Toe

Maintainability

The code is easy to trace, and most of the variables, classes, and methods names are descriptive. However, you must consider adding some comments on specific parts of the code to make it easier to understand.

In Ultimate Tic Tac Toe there were variables nx, ny, consider make all your variable names meaningful.



```
void Ultimate_Board<T>::update_summary_board(int x, int y, T symbol) {
    int nx = x / 3, ny = y / 3;
    if (is_summary_win( x: nx,  y: ny)) {
```

Consider choosing one variable to refer to one thing in the snippet you used nx and ny as x,y for the summary board but later you referred to the scale of x and scale of y as nx and ny.

Code Formatting

Code Formatting is very good there is no unnecessary whitespaces, in the other hand all whitespaces used for making the code format easier to read and trace.

Documentation

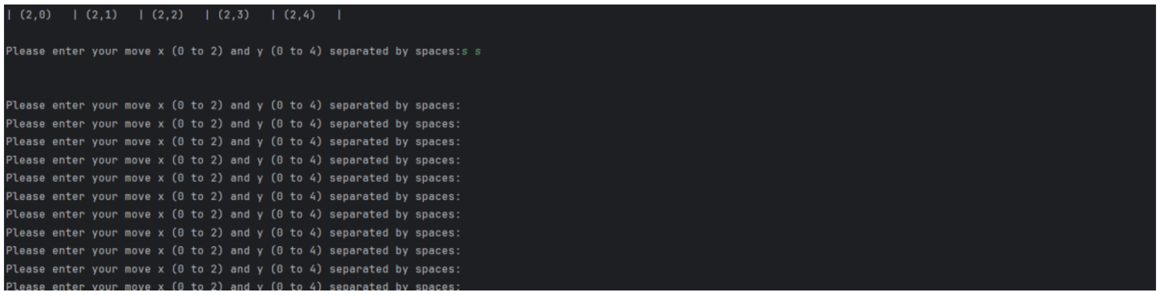
Consider making a Readme file or any documentation that explains how your program works, the algorithm you followed, release and version number, and information about the author.

Functionality

The code works well and does its job to run the game. It announces the winner when the game ends and maintains the rules of the game.

Best Practices

The single responsibility principle is followed as each function responsible for only one job as its name described and the minimal nesting used. However, you should consider handling errors that occur because of user bad inputs.



```
| (2,0)   | (2,1)   | (2,2)   | (2,3)   | (2,4)   |

Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:s s


Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
Please enter your move x (0 to 2) and y (0 to 4) separated by spaces:
```

Four in a row and Numerical & 4x4 Tic Tac Toe:

Maintainability

The code is easy to trace, and Most of the variables, classes, and methods names are descriptive. Comments included and made the code easier to understand. However, you should try to make your variable names more descriptive. Instead of names like pat, di, and dj use pattern, delta_i, delta_j.

Code Formatting

Code Formatting is very good there is no unnecessary whitespaces, in the other hand all whitespaces used for making the code format easier to read and trace as shown in the declaration of patterns.

Documentation

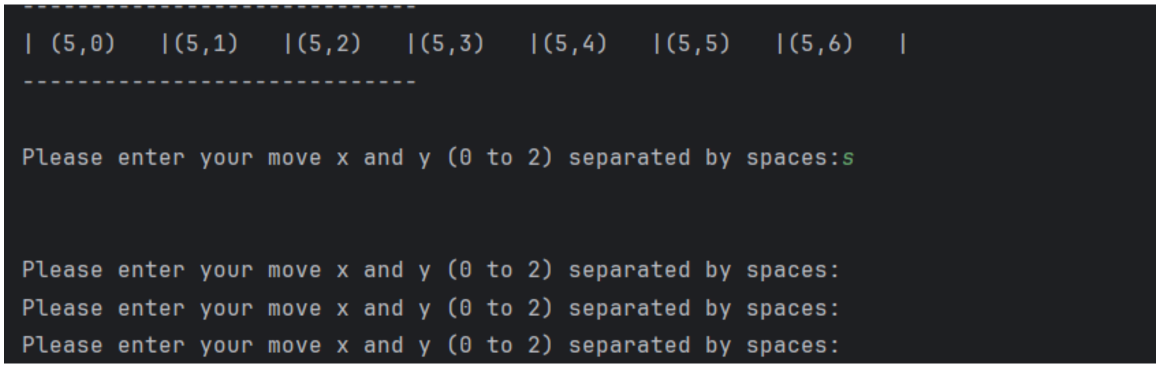
Consider making a Readme file or any documentation that explains how your program works, the algorithm you followed, release and version number, and information about the author.

Functionality

The code works well and does its job to run the game. It announces the winner when the game ends and maintains the rules of the game.

Best Practices

The single responsibility principle is followed as each function responsible for only one job as its name described and the minimal nesting used. However, you should consider handling errors that occur because of user bad inputs.

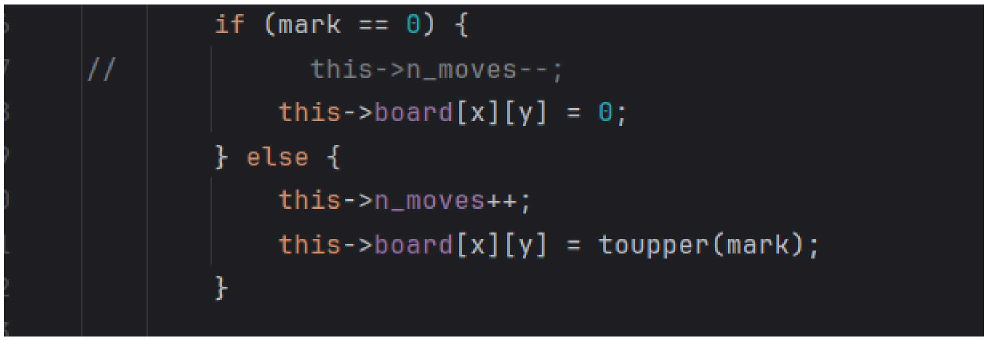


```
 ----------------------------
 | (5,0)   |(5,1)   |(5,2)   |(5,3)   |(5,4)   |(5,5)   |(5,6)   |
 ----------------------------

 Please enter your move x and y (0 to 2) separated by spaces:s


 Please enter your move x and y (0 to 2) separated by spaces:
 Please enter your move x and y (0 to 2) separated by spaces:
 Please enter your move x and y (0 to 2) separated by spaces:
```

Consider removing unnecessary comments.



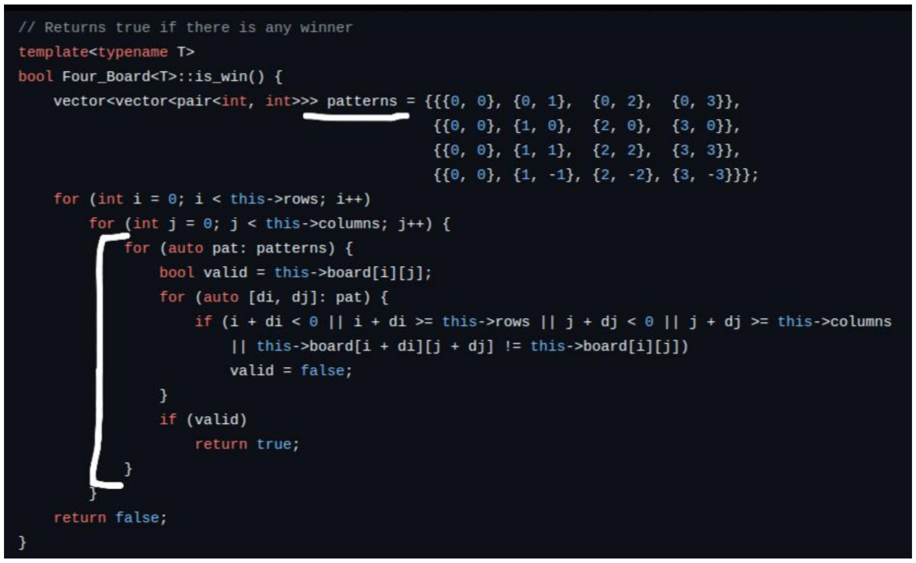
```
        if (mark == 0) {
//          this->n_moves--;
            this->board[x][y] = 0;
        } else {
            this->n_moves++;
            this->board[x][y] = toupper(mark);
        }
```

Mazen

Four-in-a-row

- The code is understandable.
- In function is_win(), in line 97, It is better handled 2 loops (1 for horizontal/vertical lines, and the other for diagonals) instead of vector<vector<pair<int,int>>> for more readability.

- There is no unnecessary whitespace in the code and the performance is good.



```cpp
// Returns true if there is any winner
template<typename T>
bool Four_Board<T>::is_win() {
    vector<vector<pair<int, int>>> patterns = {{{0, 0}, {0, 1},  {0, 2},  {0, 3}},
                                               {{0, 0}, {1, 0},  {2, 0},  {3, 0}},
                                               {{0, 0}, {1, 1},  {2, 2},  {3, 3}},
                                               {{0, 0}, {1, -1}, {2, -2}, {3, -3}}};

    for (int i = 0; i < this->rows; i++)
        for (int j = 0; j < this->columns; j++) {
            for (auto pat: patterns) {
                bool valid = this->board[i][j];
                for (auto [di, dj]: pat) {
                    if (i + di < 0 || i + di >= this->rows || j + dj < 0 || j + dj >= this->columns
                        || this->board[i + di][j + dj] != this->board[i][j])
                        valid = false;
                }
                if (valid)
                    return true;
            }
        }
    return false;
}
```

5 x 5 Tic Tac Toe

- The code is understandable.
- The function game_is_over() , in line 161, is unnecessary at all, the variable n_moves can be used to indicate whether the board is full or not, since it is incremented by 1 every time a valid move is played.
- In function getmove(), line 182, the checker part for the chosen move is better to be written in another function which return Boolean for more readability.
- There is no unnecessary whitespace in the code and the performance is good.

Numerical Tic-Tac-Toe

1. The code is understandable.
2. There is no unnecessary whitespace in the code and the performance is good.

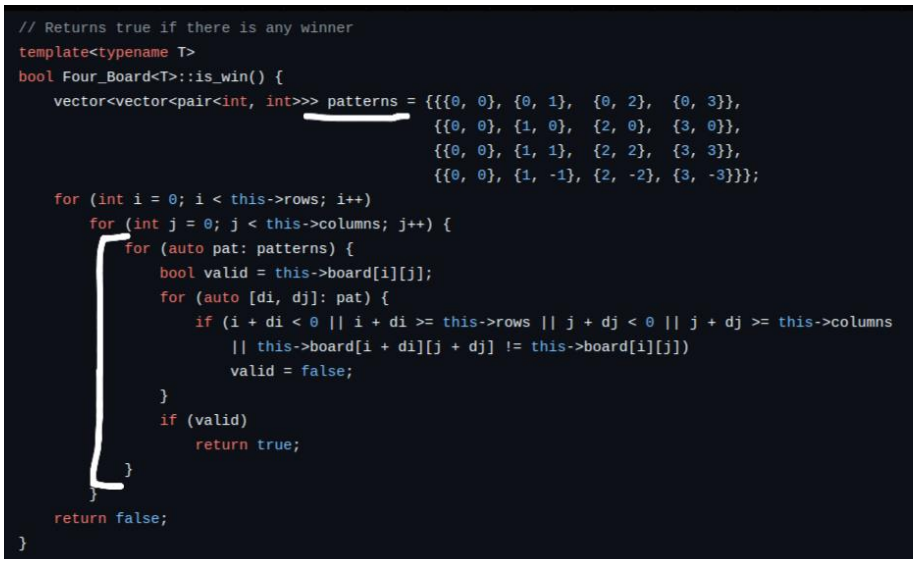
Misère Tic Tac Toe

1. The code is understandable.
2. In function getmove(), line 186, the checker part for the chosen move is better to be written in another function which return Boolean for more readability.
3. There is no unnecessary whitespace in the code and the performance is good.

4x4 Tic-Tac-Toe

1. The code is understandable.

2. In function is_win(), in line 114, It is better handled 2 loops (1 for horizontal/vertical lines, and the other for diagonals) instead of



```cpp
// Returns true if there is any winner
template<typename T>
bool Four_Board<T>::is_win() {
    vector<vector<pair<int, int>>> patterns = {{{0, 0}, {0, 1},  {0, 2},  {0, 3}},
                                               {{0, 0}, {1, 0},  {2, 0},  {3, 0}},
                                               {{0, 0}, {1, 1},  {2, 2},  {3, 3}},
                                               {{0, 0}, {1, -1}, {2, -2}, {3, -3}}};

    for (int i = 0; i < this->rows; i++)
        for (int j = 0; j < this->columns; j++) {
            for (auto pat: patterns) {
                bool valid = this->board[i][j];
                for (auto [di, dj]: pat) {
                    if (i + di < 0 || i + di >= this->rows || j + dj < 0 || j + dj >= this->columns
                        || this->board[i + di][j + dj] != this->board[i][j])
                        valid = false;
                }
                if (valid)
                    return true;
            }
        }
    return false;
}
```

vector<vector<pair<int,int>>> for more readability.

3. There is no unnecessary whitespace in the code and the performance is good.

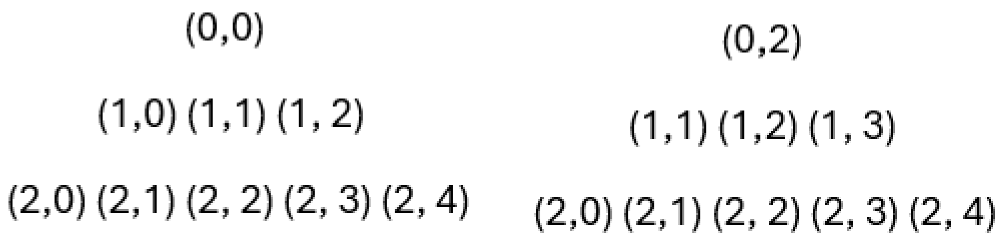
Yousef

Pyramic Tic Tac Toe

The code meets the requirements and is well-formatted. However, in my opinion, it's not very easy to read due to the indexing system. The current indexing starts with each row's first square at column 0. While this approach is functional, it can be confusing for players or readers of the code.

To improve readability and make the grid more intuitive:

- Reindex the grid so that the topmost square aligns with (0,2) instead of (0,0).



5x5 Tic Tac Toe

The code meets the requirements and is easy to read. However, there are opportunities to optimize its performance:

- In game_is_over(): Instead of counting the symbols on the board again, you could simply check if n_moves == 24 and return true.
- In getmove(): Checking boundaries is unnecessary since this is already handled in update_board().

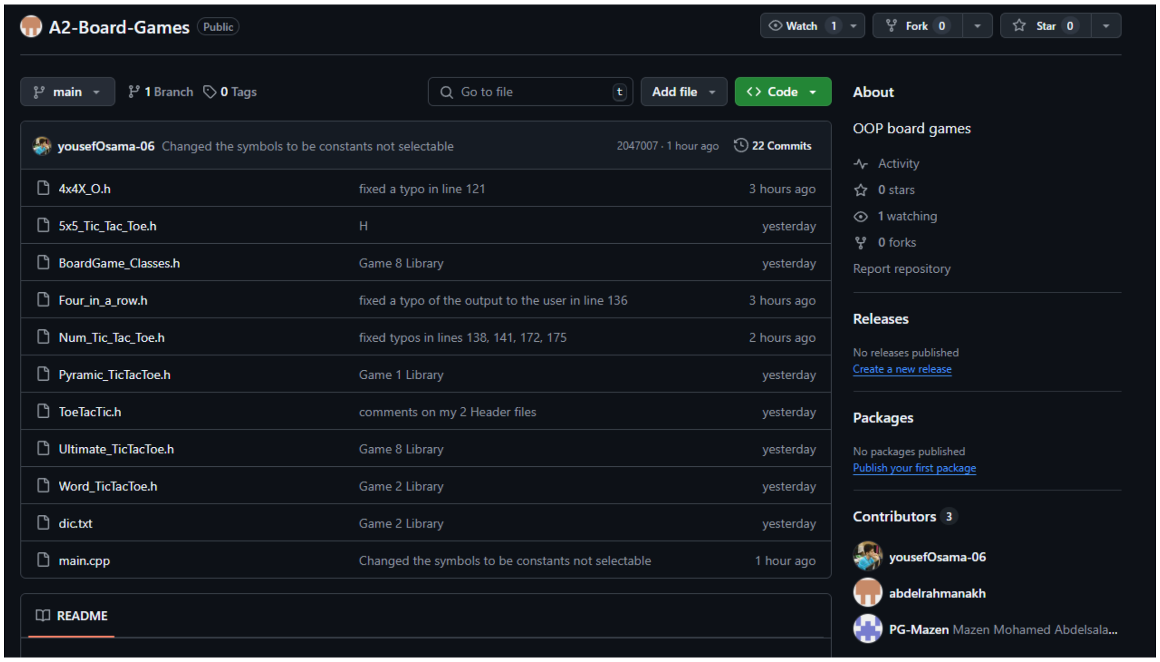
Word Tic Tac Toe

The code meets the requirements, is well-formatted, and is easy to read.

Misère Tic Tac Toe

The code meets the requirements and is well-formatted. However, the is_win() function could be implemented more efficiently and with fewer lines of code.

GitHub



A2-Board-Games Public

Watch 1 · Fork 0 · Star 0

main · 1 Branch · 0 Tags · Go to file · Add file · Code

yousefOsama-06 Changed the symbols to be constants not selectable · 2047007 · 1 hour ago · 22 Commits

4x4X_O.h	fixed a typo in line 121	3 hours ago
5x5_Tic_Tac_Toe.h	H	yesterday
BoardGame_Classes.h	Game 8 Library	yesterday
Four_in_a_row.h	fixed a typo of the output to the user in line 136	3 hours ago
Num_Tic_Tac_Toe.h	fixed typos in lines 138, 141, 172, 175	2 hours ago
Pyramic_TicTacToe.h	Game 1 Library	yesterday
ToeTacTic.h	comments on my 2 Header files	yesterday
Ultimate_TicTacToe.h	Game 8 Library	yesterday
Word_TicTacToe.h	Game 2 Library	yesterday
dic.txt	Game 2 Library	yesterday
main.cpp	Changed the symbols to be constants not selectable	1 hour ago

README

About

OOP board games

Activity
0 stars
1 watching
0 forks
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 3

yousefOsama-06
abdelrahmanakh
PG-Mazen Mazen Mohamed Abdelsala...