

**The Gaming Application**

# OOP-CS213-Assignment2\_Task2,3,4,5

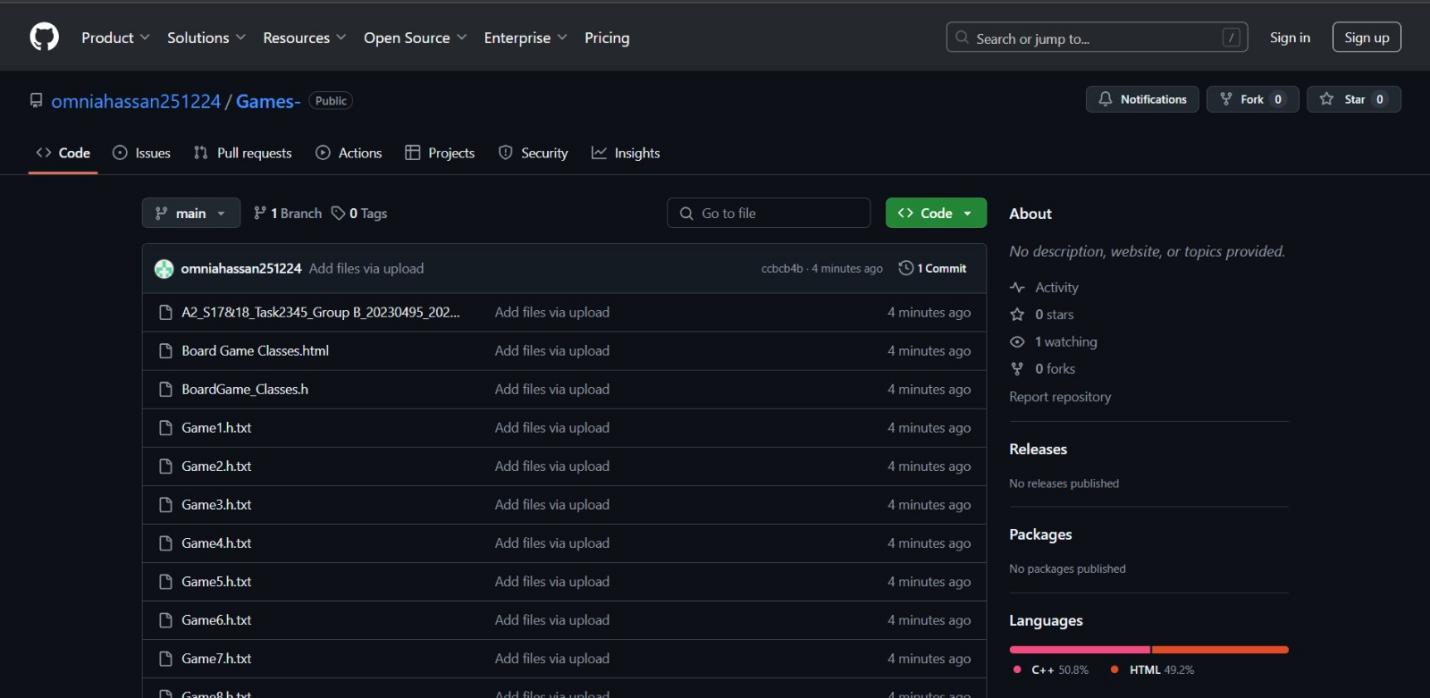
# Section: S17&18

|  |  |  |
| --- | --- | --- |
| Names | IDs | Emails |
| Nada Shaaban abdelraman | 20231187 | nadashaaban574@gmail.com |
| Omnia Hassan Sayed Hassan | 20231027 | Omniahassan251224@gmail.com |
| Youssef Hamdy Abo Elmaaty Bendary | 20230495 | 20230495@stud.fci-cu.edu.eg |

Contributions

|  |  |
| --- | --- |
| Name | Contributions |
| Youssef | Made Pyramic tic- tac-toe , word tic-tac-toe, 4\*4 tic-tac-toe  Made Human\_player , Computer player, menu , Defensive programming |
| Omnia | Made Numerical tic-tac-toe , Connect4, Ultimate tic-tac-toe  Made Human\_player , Computer player |
| Nada | Made Misere tic-tac-toe, 5\*5 tic-tac-toe  Made Human\_player , Computer player  Made report,UML |

GitHub Repository



Description

The code starts by welcoming user to program and offering him a set of choices for each game from 1 to 8. Each input is validated correctly and according to the choice of the user he plays the according game. This is done through switch case in manager function which recursively calls itself until user exits the program. Also user inputs the types and names of the two players whether man or random computer player. Each function is responsible for a certain game and uses conditions according to the options the user chose. It starts by creating an object of board type and makes a pointer of type Board that points at it. Then creates players using their constructors, sets their turns, makes an array of type Player that consists of the two players. It also makes a sharedpointer pointing the two players to the structure called SharedData in order for any changes to occur in both players. Then it makes object of gamemanager and starts the game.

The gamemanager starts by displaying board and getting moves from each player using getmove() function which is implemented in Human, computer, AI players internally by calling function checkboard() function once to see if it is a special type game with different condtions

The checkboard() function tries to dynamic cast board pointer to the special child pointer of the games (numerical, ultimate, connect4, 5\*5 , word) tic-tac-toe. Because these games have special conditions like taking a second input from user which is done in numerical, word tic-tac-toe and changing symbol of player through setsymbol() function or using special vectors for indexes or alphabet or numbers. The SharedData structure contains vectors that are needed for all games including special games and using bools in private section of human, computer player we can determine which vectors we will need and use them accordingly and since both players are pointing at same structure. Any changes will occur to both players. We used one indexing in all games to ensure ease for the users and for us.

Makemove() starts by taking index played by user and checking if it is a correct move and hasn’t been played before. According to bools it access the correct indexes vector for each game and then updates board. For ultimate it takes input at the start to see which board will be played then displays that board, the players play on it then return to the original board to check next board index. This is done through a helper class tic-tac-toe in which we create an object in

order to play a miniature version of ultimate tic-tac-toe and then take result of that game and display it on ultimate board.

All pure virtual functions inherited from board class has been implemented according to each game’s rules, board, winning conditions, etc. There were some games that needed helper functions like Misere, connect4 and ultimate tic-tac-toe.

Games

**Pyramic tic-tac-toe**

The game consists of a vector of pair of int and char where each index is represented by a character. Initialized with a value of ‘-‘and this value changes over the course of the game. Display was implemented nicely and easy for the user to see and use. Winning conditions were hard-coded.

**Connect4**

The game is implemented using a 2D vector of pairs (int, char), where each cell represents a column index and its current state. Initially, all cells are set to '-' to indicate they are empty, and their values dynamically change as the game progresses. The board's display is structured to provide a clear and intuitive visualization of the game's state, ensuring ease of use for players.

Winning conditions, including horizontal, vertical, and diagonal four-in-a-row scenarios, are explicitly hard-coded for optimal performance and accurate validation.

**Word tic-tac-toe**

The game starts by loading file dic.txt into a vector of strings. It also has a vector of pair of int and char where each index is represented by an alphabet. It has a special function getvalue() (used in AI) that returns character of chosen index. Winning condition is implemented by checking vertical, horizontal and diagonal if a word that is made is found in dic.txt then it is a win for that player. Display was implemented nicely and easy for the user to see and use.

**Numerical\_Board**

The game is implemented using a 1D vector of pairs (int, char) to represent a 3x3 board. Each cell is initialized with an index (0 to 8) and the value '-', which changes during gameplay. The board display is structured to include separators for rows and columns, providing a clear and user-friendly visualization. Winning conditions are hard-coded, checking if the sum of specific cell values equals 15, and the logic is optimized for accurate and efficient validation.

**Misere Ti-Tac-Toe**

**A game where the loser is the first to complete a row, column, or diagonal with three consecutive marks. The main objective is to avoid completing any line while trying to force the opponent to do so. The game follows the same basic rules (a 3×3 grid and alternating turns for each player) but requires different strategies, focusing on spreading marks and avoiding traps that could lead to a loss.**

**5\*5 Tic-Tac-Toe**

**A game aimed for align five consecutive marks, either horizontally, vertically, or diagonally, to win the game. The larger grid increases the complexity and strategic depth of gameplay, as players must plan their moves carefully to block their opponent while creating their own opportunities to win. This variation offers a more challenging and engaging experience compared to the classic version.**

**Tic-Tac-Toe**

A helper class that is used in ultimate tic-tac-toe to facilitate displaying boards and getting win from either player or a draw and displaying it on the main ultimate board.

**Ultimate Tic-Tac-Toe**

Similar to normal tic-tac-toe with a twist that each index represents a board on its own and winning that board means putting your symbol on that board. The game consists of a vector of pair of int and char where each index is represented by a character. Initialized with a value of ‘-‘and this value changes over the course of the game. Display was implemented nicely and easy for the user to see and use. Winning conditions were hard-coded.

Code Quality and review

First issue encountered by Youssef is that he made a gamemanager separate from the original gamemanager provided. It was a lot better and less complicated to deal with but the doctor refused it. So he made sure to use original gamemanager which proved to be a challenge to link everything together and changing entire structure of code. SharedData had some problems when we used vectors other than the the main vector indexes but was resolved using conditions and bools. In Ultimate tic-tac-toe there was an issue when two players not of same type played against each other, this was resolved using a special method in tic-tac-toe class using conditions and bool special. There was also an issue with saving progress of game when initially trying to make an object of tic-tac-toe in Ultimate tic-tac toe which was resolved by making the object in the player class itself.

Second issue was encountered by Omnia after Youssef made new structure for the code where the numerical and connect 4 weren’t operating correctly. Connect 4 had a problem with update board where it would take infinite input and wasn’t able to update board which was solved by making it return true always. Numerical had a problem where it tried to access elements out of bounds in updateboard which was resolved also. Mai also faced a problem with making the AI for her games which is that the AI played played and kept playing until it won not giving the other player a chance to play.

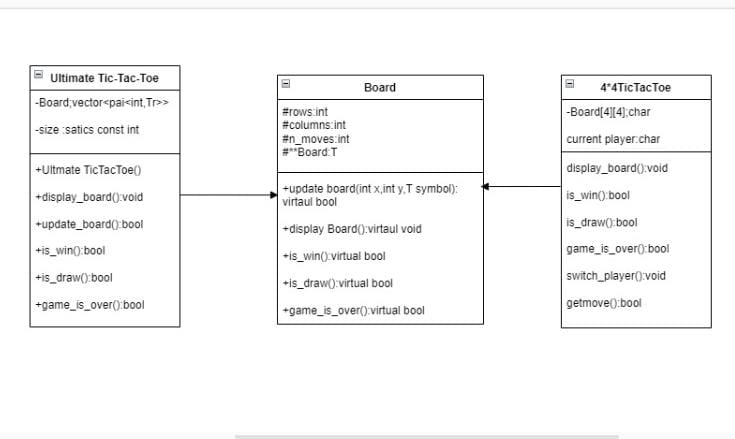
Third issue was encountered by Nada in the misere which had a problem with checking the winner. We had to use a hidden bool and special function in order to make a virtual turn and switch to the correct winner

Also Ultimate tic-tac-toe proved to be quite challenging. It was hard to implement each index having its own board but we managed to do it.

Overall code’s problem is trying to understand how getmove works in Human, Computer and AI as it is pretty difficult to follow everything that is happening and so we made a meeting in order to explain the code for each other which proved to be quite helpful.

The requirements have been met. All games work smoothly, all validations are correct. The code is close to simple as much as possible. The code is effiecient and the performance is really good. All games are displayed nicely and easy for the user to understand and play. No chance for errors or problems in the code.

UML



### GUI

**Tools Used:**

•Qt Creator: Qt Creator is an integrated development environment (IDE) designed for writing and developing project code. It offers a powerful environment that supports programming in C++ and enables the development of graphical user interfaces (GUIs) using Qt Widgets. Qt Creator is an effective tool that streamlines the coding, debugging, and testing processes of applications.

•Qt Widgets: Qt Widgets was utilized to create traditional graphical user interfaces (GUIs) that rely on standard elements such as buttons, text fields, and drop-down lists. Qt Widgets provides robust tools for building interactive and flexible interfaces, facilitating the development of user-friendly and efficient applications.

**GUI Link:**

https://drive.google.com/drive/folders/195xvK9E3HZzBBONj4TMrzWzlBiA1wWky