Cairo University

Faculty of Computers and Artificial Intelligence

CS213 - Object Oriented Programming

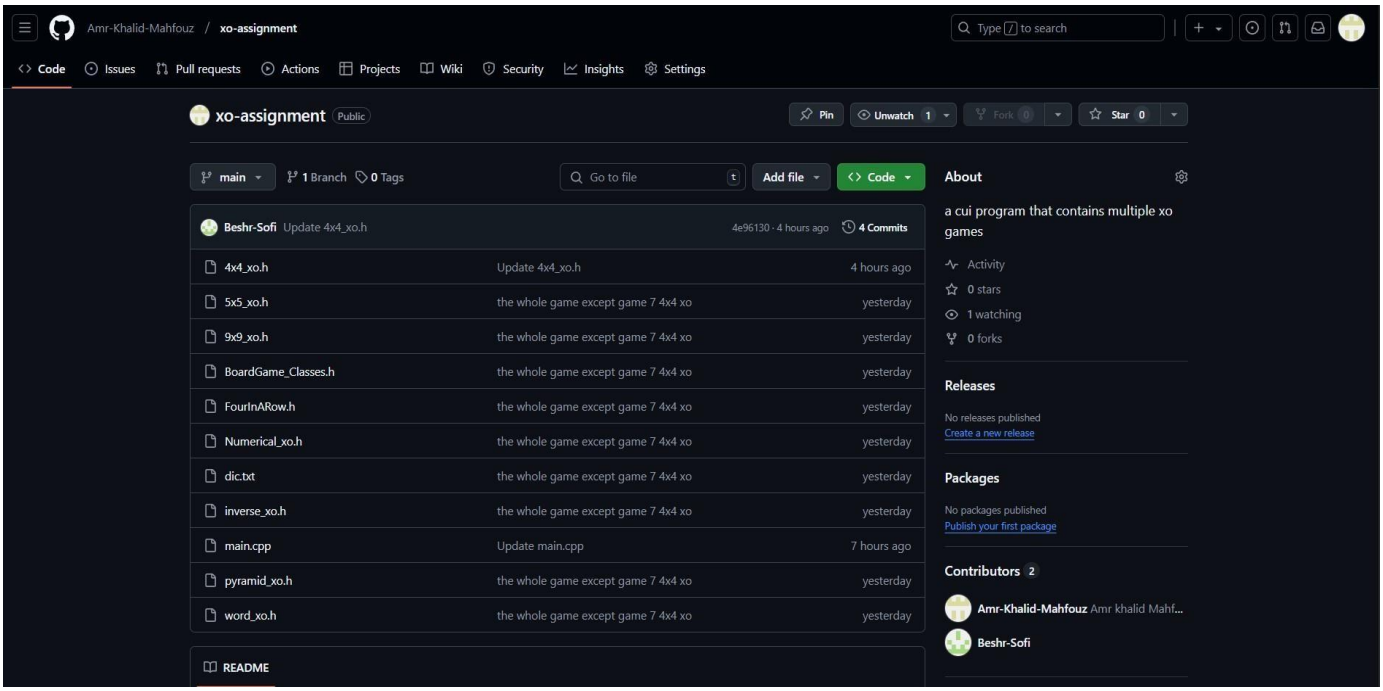
Task 2, 3

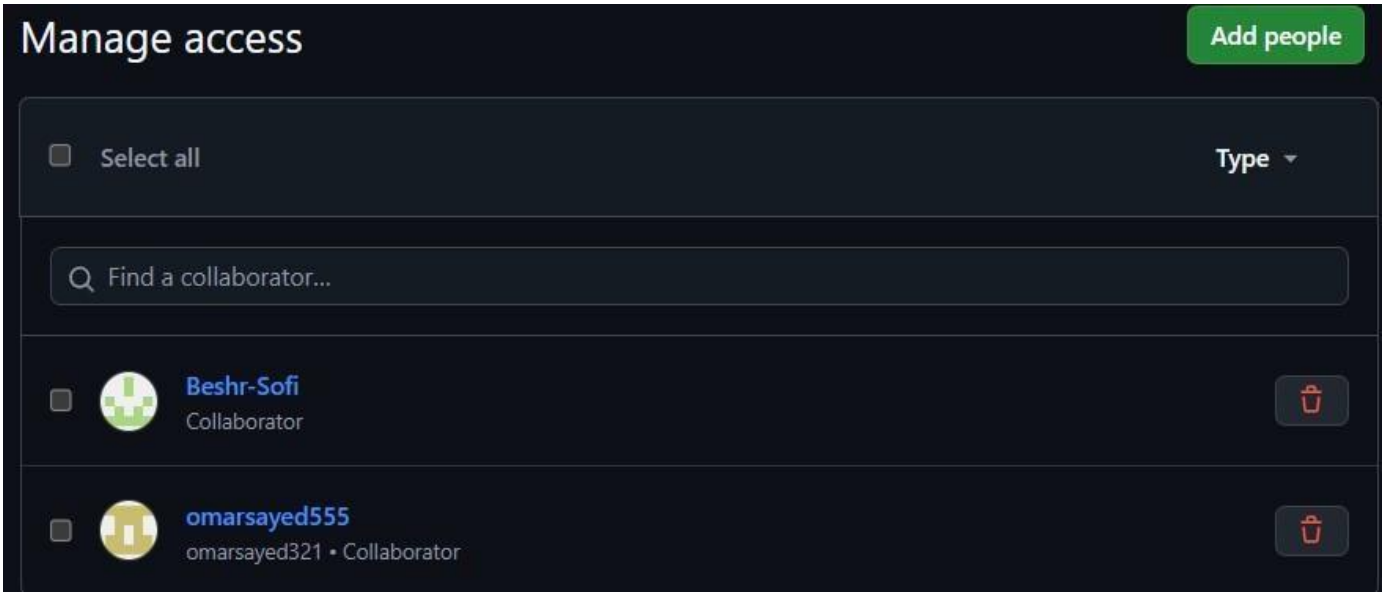
Name	ID	Section
Omar Sayed Soliman Omarsayedsoliman165@gmail.com	20230256	S26
Amr Khalid Mahfouz Moora99kh@gmail.com	20230272	S6
Mohamed Beshr Mohamed Beshrsofi2005@gmail.com	20230717	S6

Under the Supervision of **DR.Mohamed El-Ramly**

Name	What he did
Omar Sayed Soliman	Games 2 & 5
Amr khalid Mahfouz	Games 1, 4 & 8
Mohamed Beshr Alsofi	Games 3 &, 6 & 7

GitHub repository: [Amr-Khalid-Mahfouz/xo-assignment: a cui program that contains multiple xo games]





Game descriptions

Game 1 Pyramid Tic Tac Toe:

1. pyramid_xo_board Class:
    - Represents the pyramid-shaped board with 3 rows and 5 columns.
    - The constructor initializes an empty board.
    - The update_board function updates the board with the player's symbol at a valid position.
    - The display_board function displays the current state of the board in a pyramid-like format.
    - The is_win function checks if a player has won by matching symbols in rows, columns, or diagonals.
    - The is_draw function checks if the game is a draw (the board is full and there is no winner).
    - The game_is_over function checks if the game is finished (either win or draw).
2. pyramid_xo_player Class:
    - Represents a human player who enters their move.
    - The getmove function asks the player for their move's coordinates (x and y).
3. pyramid_random_xo_player Class:
    - Represents a random player who chooses a random move.

Game 2 Four in a Row:

1. FourInARow_Board

. A template class derived from Board<T>

. Implements a 6x7 board for the Four-in-a-Row game.

. Key Methods:

- update_board(int x, int y, T symbol): Places a symbol in the lowest available square in the specified column.
- display_board(): Displays the board with current game state.
- is_win(): Checks for win conditions (horizontal, vertical, and diagonal alignments).
- is_draw(): Checks if the game has reached a draw.
- game_is_over(): Combines is_win() and is_draw() to determine if the game has ended.

2. FourInARow_Player

- A template class derived from Player<T>.
- Represents a human player for Four-in-a-Row.

Key Methods:

- getmove(int& x, int& y): Prompts the player to input their move (column only).

3. FourInARow_Random_Player

- A template class derived from RandomPlayer<T>.
- Represents an AI player that makes random moves.

Key Methods:

- getmove(int& x, int& y): Randomly selects a column for the move.

Game 3 5x5 Tic Tac Toe:

Main Class: xo_5x5_board

1. Purpose: Represents the 5x5 tic-tac-toe game board, managing game logic, player states, and scoring.
2. Key Data Members:
    - names[2]: Stores pointers to two players. o FirstPlayer & SecondPlayer: Boolean vectors tracking the moves of each player. o FirstPlayerPoints & SecondPlayerPoints: Scores for each player. o board: A 5x5 matrix representing the game state.
    - n_moves: Counter for the number of moves made.
3. Constructor: Initializes the board, player data, and scoring mechanisms.
4. Game Logic Methods:
    - update_board: Updates the board with the player's move if valid. o display_board: Displays the board state with player marks or coordinates for empty spots.
    - is_win: Checks if a player has won by forming specific patterns (horizontal, vertical, diagonal). o is_draw: Determines if the game ends in a draw (equal scores with a full board).
    - game_is_over: Determines if the game is over, announcing the winner if applicable.

Player Classes

1. xo_5x5_player:
    - Represents a human player.
    - Prompts the user for their move coordinates (x and y).
2. random_xo_5x5_player:
    - Represents an AI or computer player making random moves.
    - Automatically generates valid random moves.

Game 4 word Tic Tac Toe:

1. word_x_o_board Class:

    - Member variables:

        o three_letter_words: A vector that contains a list of three-letter words, it is loaded from a file (dic.txt).

    - Board Setup: The board is a 3x3 grid, initialized to empty.
    - The update_board function places a letter on the board at position (x, y) if it's empty and valid (letter from A-Z).
    - The display_board function displays the current state of the board, showing either the placed letters or the empty coordinates.
    - The is_win function checks if a valid three-letter word is formed in any row, column, or diagonal by concatenating the letters and searching for them in the three_letter_words list.
    - The is_draw function checks if the game is a draw (board full, no winner).
    - The game_is_over function checks if the game is over due to a win or a draw.

2. word_x_o_player Class:

    - Represents a human player.
    - getmove Function: Prompts the player to enter a letter and its coordinates (x, y) on the board. The entered letter is converted to uppercase.

3. random_word_x_o_player Class:

    - A class that represents a random computer player for the Word XO game.
    - getmove Function: Randomly selects a letter (from A-Z) and a position (x, y) on the board to make a move.

Game 5 Numerical Tic-Tac-Toe:

1. NumericalBoard

- A class derived from Board<int>.
- Implements a 3x3 board for Numerical Tic-Tac-Toe.

Key Methods:

- update_board(int x, int y, int number): Places a number on the board if the cell is empty and the position is valid.
- display_board(): Displays the current state of the board.
- is_win(): Checks for a win condition where the sum of numbers in any row, column, or diagonal equals 15.
- is_draw(): Checks if the game has reached a draw state.
- game_is_over(): Combines is_win() and is_draw() to determine if the game has ended.

2. NumericalPlayer

- A class derived from Player<int>.
- Represents a player in Numerical Tic-Tac-Toe.

Key Attributes:

- unordered_set<int> available_numbers: Tracks the numbers that the player can use.

Key Methods:

- getmove(int& x, int& y): Prompts the player to enter their move (row, column, and number), ensuring it is valid and available.

Game 6 Misere Tic Tac Toe:

Main Class: Inverse_X_O_Board

1. Purpose: Represents the game board for a 3x3 "Inverse Tic-Tac-Toe" game, where the rules or mechanics may differ from standard tic-tac-toe.
2. Key Data Members:
    - Inherits from the Board<T> class, making it compatible with generic board game frameworks. o board: A dynamically allocated 3x3 matrix for game state storage.
    - n_moves: Tracks the number of moves made.
3. Constructor:
    - Initializes a 3x3 board with empty cells (0), setting up the game environment.
4. Game Logic Methods:
    - update_board: Validates and applies a player's move or undoes it.
    - display_board: Outputs the current board state, showing player marks or available coordinates.
    - is_win: Checks for a winning condition (three identical non-empty marks in a row, column, or diagonal).
    - is_draw: Determines if the game ends in a draw (all moves completed without a winner).
    - game_is_over: Indicates whether the game has ended (either a win or draw). Player Classes
1. Inverse_X_O_Player:
    - Represents a human player.
    - Prompts for and processes the player's move (x and y coordinates).
2. Inverse_X_O_Random_Player:
    - Represents a computer-controlled player making random moves.
    - Automatically generates random valid moves.

Game 7 4x4 Tic-Tac-Toe

Main Class: xo_4x4_board

1. Purpose: Represents the game board for a 4x4 tic-tac-toe game with modified rules and player interaction mechanics. 2. Key Data Members:

- Inherits from the Board<T> class. o names[2]: Stores pointers to two players. o board: A dynamically allocated 4x4 matrix for game state storage.
- n_moves: Tracks the number of moves made.

3. Constructor:

- Initializes the board as a 4x4 grid with preset positions for 'X' and 'O' pieces. o Sets the initial state for the two players.

4. Game Logic Methods:

- update_board: Validates and processes a player's move, including restrictions based on the current state and turn.
- display_board: Outputs the current state of the board with player pieces or coordinates for empty cells.
- is_win: Checks for a winning condition (three consecutive identical non-empty marks in rows, columns, or diagonals).
- is_draw: returns false (draw logic not implemented in this game mode). o game_is_over: Determines whether the game ends with a win condition. Player Classes

1. xo_4x4_player:
   - Represents a human player.
   - Prompts for and processes the player’s choice of piece to move and its target position.
2. xo_4x4_random:
   - Represents a computer-controlled player that makes random moves.
   - Automatically generates random valid moves based on the current board state.

Game 8 9x9 Tic Tac Toe:

1. xo_9x9_board:

- This class represents a 9x9 XO game board, consisting of 9 smaller 3x3 boards.
- Member Variables:
    - player1_wins and player2_wins: Sets that track the indices of the 3x3 boards that each player has won.
    - done_boards: A Boolean array of size 9, indicates whether a particular 3x3 board has been completed (a win is detected).
- Constructor: Initializes the board to 9 rows and 9 columns, setting each cell to 0 (empty), and the number of moves (n_moves) to 0.
- The check_all_boards function checks every 3x3 sub-board for a win condition (horizontal, vertical, or diagonal). If a win is found, it marks that sub-board as "done" and adds the winning board's index to the corresponding player's set.
- The update_board function Updates a specific cell on the main board with a player's symbol if the cell is empty. After the move, it calls check_all_boards function.
- The fill_the_board function fills the 3x3 sub-boards of the main board with the respective player's symbol ('X' or 'O') if that player has won in the sub-board.
- The display_board function displays the current state of the board.
- The add_to_set function adds the index of a winning 3x3 sub-board to the appropriate player's set of wins.
- The find_set function checks if a given set of sub-board indices corresponds to a win for the given player.
- The is_win function determines if either player has won the game by checking if any of the winning combinations (of sub-board indices) are entirely in a player's winning set.
- The is_draw function Returns true if the game is a draw, meaning all moves are made but no one has won.
- The game_is_over function returns true if the game has either ended in a win for any of the players or a draw.

2. xo_9x9_player:

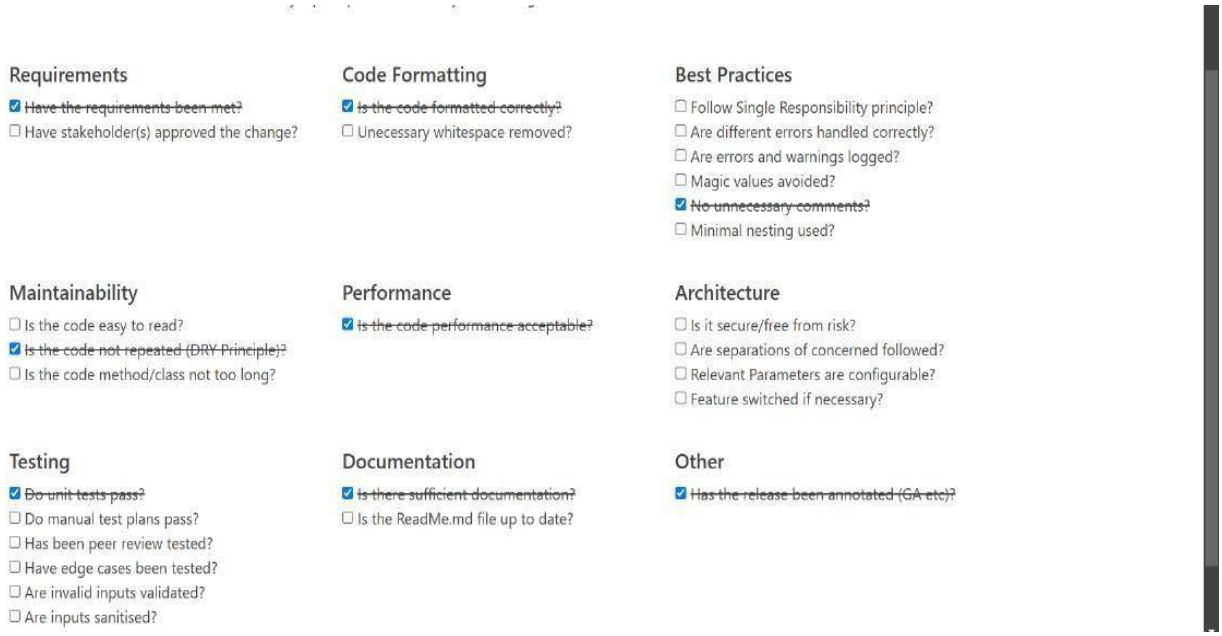
- A class representing a human player in the 9x9 Tic-Tac-Toe game.
- The getmove function prompts the user to enter their move (coordinates x and y), where both x and y are integers between 0 and 8.

3. random_xo_9x9_player:

- A class representing a random AI player in the 9x9 Tic-Tac-Toe game.
- The getmove function chooses random x and y coordinates for the move between 0 and 8

Code Review

Mohammed’s Review of Omar’s Code:



Requirements
- [x] ~~Have the requirements been met?~~
- [ ] Have stakeholder(s) approved the change?

Code Formatting
- [x] ~~Is the code formatted correctly?~~
- [ ] Unecessary whitespace removed?

Best Practices
- [ ] Follow Single Responsibility principle?
- [ ] Are different errors handled correctly?
- [ ] Are errors and warnings logged?
- [ ] Magic values avoided?
- [x] ~~No unnecessary comments?~~
- [ ] Minimal nesting used?

Maintainability
- [ ] Is the code easy to read?
- [x] ~~Is the code not repeated (DRY Principle)?~~
- [ ] Is the code method/class not too long?

Performance
- [x] ~~Is the code performance acceptable?~~

Architecture
- [ ] Is it secure/free from risk?
- [ ] Are separations of concerned followed?
- [ ] Relevant Parameters are configurable?
- [ ] Feature switched if necessary?

Testing
- [x] ~~Do unit tests pass?~~
- [ ] Do manual test plans pass?
- [ ] Has been peer review tested?
- [ ] Have edge cases been tested?
- [ ] Are invalid inputs validated?
- [ ] Are inputs sanitised?

Documentation
- [x] ~~Is there sufficient documentation?~~
- [ ] Is the ReadMe.md file up to date?

Other
- [x] ~~Has the release been annotated (GA etc)?~~

Amr’s review of Mohammed’s code:

Requirements
- [x] ~~Have the requirements been met?~~
- [ ] Have stakeholder(s) approved the change?

Code Formatting
- [x] ~~Is the code formatted correctly?~~
- [ ] Unecessary whitespace removed?

Best Practices
- [ ] Follow Single Responsibility principle?
- [x] ~~Are different errors handled correctly?~~
- [x] ~~Are errors and warnings logged?~~
- [x] ~~Magic values avoided?~~
- [x] ~~No unnecessary comments?~~
- [ ] Minimal nesting used?

Maintainability
- [x] ~~Is the code easy to read?~~
- [ ] Is the code not repeated (DRY Principle)?
- [x] ~~Is the code method/class not too long?~~

Performance
- [x] ~~Is the code performance acceptable?~~

Architecture
- [ ] Is it secure/free from risk?
- [ ] Are separations of concerned followed?
- [ ] Relevant Parameters are configurable?
- [ ] Feature switched if necessary?

Testing
- [x] ~~Do unit tests pass?~~
- [x] ~~Do manual test plans pass?~~
- [ ] Has been peer review tested?
- [ ] Have edge cases been tested?
- [x] ~~Are invalid inputs validated?~~
- [x] ~~Are inputs sanitised?~~

Documentation
- [x] ~~Is there sufficient documentation?~~
- [x] ~~Is the ReadMe.md file up to date?~~

Other
- [ ] Has the release been annotated (GA etc)?

Omar’s review of Amr’s code: