Faculty of Computers and Artificial Intelligence

Cairo University

OOP

Dr. Mohamed EL_Ramly

Assignment 2 Task 2,3,4,5

CAIRO UNIVERSITY

Name:	IDS:	Emails:	Section:
Marwan Tamer Sayed Ali	**20230380**	**marwantamer004@gmail.com**	**S19**
Mohamed Refaat Mohamed Awd	**20230337**	**mrefaat8853@gmail.com**	**S19**
Rana Tarek Ahmed Fouad Ibrahim	**20230632**	**r.tarekahmed@gmail.com**	**S20**

Name:	Tasks:
Marwan Tamer Sayed Ali	**Game 2, 5 \ Main function**
Mohamed Refaat Mohamed Awd	**Game 1, 4, 8**
Rana Tarek Ahmed Fouad Ibrahim	**Game 3, 6, 7**

1. Pyramid Game

PyramidBoard (Template Class):

Description: Represents a pyramid-shaped game board, inherited from the Board class. The board has 3 rows with increasing columns in each row (1, 3, 5).

Attributes:

- rows and columns define the board dimensions.
- board is a 2D array that stores the game state (characters).
- n_moves keeps track of the number of moves made.

Methods:

- update_board(int x, int y, T symbol): Updates the board with a symbol at position (x, y).
- display_board(): Displays the board in a pyramid format.
- is_win(): Checks for a winning condition (horizontal, vertical, or diagonal).
- is_draw(): Checks if the game is a draw (no winner and all cells filled).
- game_is_over(): Returns true if the game is either won or drawn.

X_O_Player (Template Class):

- **Description**: Represents a human player in the Pyramid game.
- **Attributes**: Inherits from Player<T>, storing the player's name and symbol.
- **Methods**:
    - getmove(int& x, int& y): Prompts the user to enter the coordinates for the next move.

X_O_Random_Player (Template Class):

- **Description**: Represents an AI player that selects random moves.
- **Attributes**: Inherits from RandomPlayer<T>, stores the player's symbol and name.
- **Methods**:
    - getmove(int& x, int& y): Randomly selects an available position on the board.

Suggested Improvements:

UI Enhancement: Improve the user interface by making the board display clearer. Currently, it's hard to see where the columns and rows are. Consider using symbols or a more intuitive format

---

2. **Four in a Row Game** (Connect Four)

FourInARowBoard (Template Class):

- **Description**: Represents the game board for the "Four In a Row" game, inheriting from the Board class. The board has 6 rows and 7 columns, and the objective is to connect four symbols in a row.
- **Attributes**:
    - rows and columns define the board dimensions.

- board is a 2D array representing the state of each cell ('.' for empty cells).
- n_moves keeps track of the number of moves made.

- **Methods**:
  - get_first_empty_row(int col): Returns the first empty row in the given column.
  - update_board(int x, int y, T symbol): Updates the board with a symbol in the first available row of the specified column.
  - display_board(): Displays the board.
  - is_win(): Checks if there's a winner by checking horizontal, vertical, and diagonal lines.
  - is_draw(): Checks if the game is a draw (no winner and all cells filled).
  - game_is_over(): Returns true if the game is either won or drawn.
- **Friend Class**: FourInARowMinimaxPlayer<T> is declared as a friend, meaning it can access private and protected members of this class.

 FourInRowPlayer (Template Class):

- **Description**: Represents a human player in the "Four In a Row" game.
- **Attributes**: Inherits from Player<T>, stores the player's name and symbol.
- **Methods**:
  - getmove(int& x, int& y): Prompts the player to enter a column number where they want to drop their symbol. The row is determined by the first available space in that column.

Suggested Improvements:

- **Column Full Validation:** Add a more intuitive method of notifying the player when a column is full. This can be done with a message after a player attempts an invalid move.
- **Move Preview**: Let the player see a preview of where their move will land before confirming the move, which enhances usability and reduces user frustration.

3.X_O_Board5X5 Game

1. X_O_Board5X5:

This class represents the game board for a 5x5 grid-based game of Tic-Tac-Toe (X and O). It inherits from the Board<T> class and provides methods to:

- **update_board(int x, int y, T symbol)**: Updates the board at position (x, y) with the given symbol if the position is empty.
- **display_board()**: Displays the current state of the board.
- **is_win()**: Checks if there is a winner by counting three-in-a-row combinations for each player ('X' and 'O').
- **is_draw()**: Checks if the game is a draw (if the number of three-in-a-row combinations for both players is the same and all moves are made).
- **game_is_over()**: Returns true if the game is over (either a win or draw), otherwise false.
- **count_threes(T symbol)**: Helper function that counts the number of three-in-a-row combinations for a given symbol (either 'X' or 'O'). It checks rows, columns, and diagonals.

2. X_O_HumanPlayer:

This class represents a human player in the game. It inherits from the Player<T> class and provides a method to get the player's move:

- **getmove(int& x, int& y)**: Prompts the player to enter a row and column for their move.

3. X_O_RandomPlayer:

This class represents a random AI player. It inherits from the RandomPlayer<T> class and provides a method to generate a random move:

- **getmove(int& x, int& y)**: Generates a random move by selecting a random row and column.

Suggested Improvements:

Improvement: The display could be simplified and made more user-friendly

---

4. Word Tic-Tac-Toe Game:

1. **WordBoard Class:**
    - This class inherits from the Board class and represents the game board for a word-based game.
    - It has the following key functionalities:
        - update_board(int x, int y, T mark): Updates the board at position (x, y) with a symbol mark, checking if the move is valid.
        - display_board(): Displays the current state of the board, including the positions and marks.
        - is_win(): Checks if there is a winning combination of characters on the board.
        - is_draw(): Checks if the game has ended in a draw (i.e., all cells are filled and there is no winner).
        - game_is_over(): Checks if the game is over (either through a win or a draw).
        - isWordInFile(const string& word): Verifies whether a word exists in the dictionary file "dic.txt".
2. **WordPlayer Class:**
    - This class represents a human player who participates in the word-based game. It inherits from the Player class.
    - Key functionalities include:
        - getmove(int& x, int& y): Prompts the player to enter a move (row and column).
        - getmove(int& x, int& y, char& c): Prompts the player to enter a move with both coordinates and a character.
3. **Word_Random_Player Class:**
    - This class represents a computer-controlled player that makes random moves. It inherits from the RandomPlayer class.
    - Key functionalities include:
        - getmove(int& x, int& y): Chooses a random move (coordinates).
        - getmove(int& x, int& y, char& c): Chooses a random move (coordinates and character).

Suggest improvement:

---

5. Numerical Tic-Tac-Toe Game

NumericalBoard Class:

- This class defines the board for the game. It has methods for updating the board (update_board), displaying the board (display_board), checking for a win (is_win), checking for a draw (is_draw), and determining if the game is over (game_is_over).
- The check_sum method is used to determine if three numbers sum to 15, which is the winning condition (like the magic square rule).
- The update_board method checks if the move is valid (in bounds, not already occupied) and updates the board with the player's symbol.

HumanPlayer Class:

- This class represents a human player in the game. The getmove method allows the player to select a number and make a move (choosing an empty cell on the board).
- The available numbers are displayed for the player, and the player can choose a valid number that hasn't been used already.
- The method validates the move and updates the board with the selected number, ensuring the move is legal and the number hasn't been used globally.

NumericalRandomPlayer Class:

- This class represents a computer player that selects random valid moves from the available numbers and positions.
- It follows a similar approach as the human player but without player input, randomly selecting an available move.

Global State:

- globalUsedPositions tracks all used positions on the board.
- globalUsedNumbers tracks all numbers that have been used by any player.

Suggested Improvements:

Display Enhancements: The display board method currently prints the board in a readable way. Consider adding more features like highlighting the winning combination.

---

6. X_O_Board_misere Game

X_O_Board_misere Class

- **Constructor:** Initializes a 3x3 board with all positions set to 0 (empty).
- **update_board(int x, int y, T mark):** Updates the board with the player's mark at the specified position. If the player chooses to undo the move (mark = 0), it decrements the move count and clears the board position.
- **display_board():** Displays the current state of the board, showing the coordinates and marks in the cells.
- **is_win():** Checks whether there's a winning combination of three marks in a row, column, or diagonal.
- **is_draw():** Returns true if all positions are filled and there's no winner.
- **game_is_over():** Returns true if the game is over, either due to a win or a draw.

X_O_Player_misere Class

- **Constructor:** Initializes a player with a name and a mark (symbol).

- **getmove(int& x, int& y):** Prompts the player to enter the coordinates for their next move.

X_O_Random_Player_misere Class

- **Constructor:** Initializes a random player, sets the dimension of the board to 3, and gives the player the name "Random Computer Player".
- **getmove(int& x, int& y):** Randomly selects coordinates between 0 and 2 for the next move.

Suggested Improvements:

--

---

7. 4x4 Tic-Tac-Toe Game:

Board Class (X_O_Board<T>):

- Represents a 4x4 grid.
- Initializes with some pre-filled tokens (X and O) at specific positions.
- Enforces rules for token movement, including adjacency and ownership validation.
- Supports win, draw, and game-over checks.
- Offers a display function to print the board's current state.

Human Player Class (X_O_Human_Player<T>):

- Allows a human player to select a token and move it to a valid adjacent position.
- Validates input thoroughly to ensure moves comply with the game's rules.

Random Player Class (X_O_Game_Random_Player<T>):

- Implements a random AI player that moves a token to a valid adjacent position randomly.

Template Design:

- Makes the game generic and adaptable to different types of tokens (char, int, etc.).

Rules:

- Movement is limited to adjacent cells, as checked by is_adjacent().
- Each player can only move their own tokens.
- Moves are restricted to empty cells.

Win Conditions:

- The code checks for 3 consecutive identical symbols horizontally, vertically, or diagonally.
- The win logic assumes a traditional Tic-Tac-Toe-like pattern, adjusted for the 4x4 grid.

Draw Conditions:

- A draw occurs when all cells are filled without a win condition being met.

Human Player Input:

- Input is robustly validated, ensuring the chosen token belongs to the player and the move is within valid bounds.

Random Player Logic:

- Randomly picks a token and destination, ensuring they follow adjacency and empty cell rules.

Suggestions for Improvement:

Initialization: The X_O_Board constructor hardcodes initial tokens (X and O). This design assumes a specific starting layout, which may limit flexibility for different setups.

---

8. Ultimate Tic Tac Toe Game:

Board Class (Small_XO_Board<T>)

- Represents a 3x3 grid for a single Tic Tac Toe board.
- Initialization:
  - Creates an empty 3x3 board.
  - Tracks the winning symbol (X or O) when a win occurs.
- Rules:
  - Supports token placement with update_board(), ensuring moves are within bounds.
  - Enforces win conditions (3 in a row horizontally, vertically, or diagonally).
  - Checks for a draw when all cells are filled without a winner.
- Game Status:
  - Provides methods (is_win(), is_draw(), game_is_over()) to validate the current game state.
- Display:
  - Offers display_board() to show the current board layout with symbols or indices for empty cells.

Ultimate Board Class (Ultimate_XO_Board<T>)

- Represents a 3x3 grid of Small_XO_Board instances, forming a 9x9 ultimate Tic Tac Toe game.
- Initialization:
  - Dynamically allocates and initializes 9 Small_XO_Board objects.
- Rules:
  - Token placement is mapped to the appropriate small board, ensuring valid moves.
  - Win conditions are checked for the 3x3 grid of small boards, based on which player wins the majority of boards.
  - Draws occur when all small boards are either won or drawn, and no overall winner exists.
- Game Status:
  - Combines statuses from small boards to determine the overall game state (win, draw, or ongoing).
- Display:
  - Provides a comprehensive view of the entire game state, including individual small boards.

Human Player Class (Ultimate_Player<T>)

- Allows a human player to make moves.
- Rules:
  - Validates input to ensure the selected cell belongs to the current player and is within bounds.
- Interaction:
  - Guides the user to select a valid move, displaying error messages for invalid input.

Random Player Class (Ultimate_Random_Player<T>)

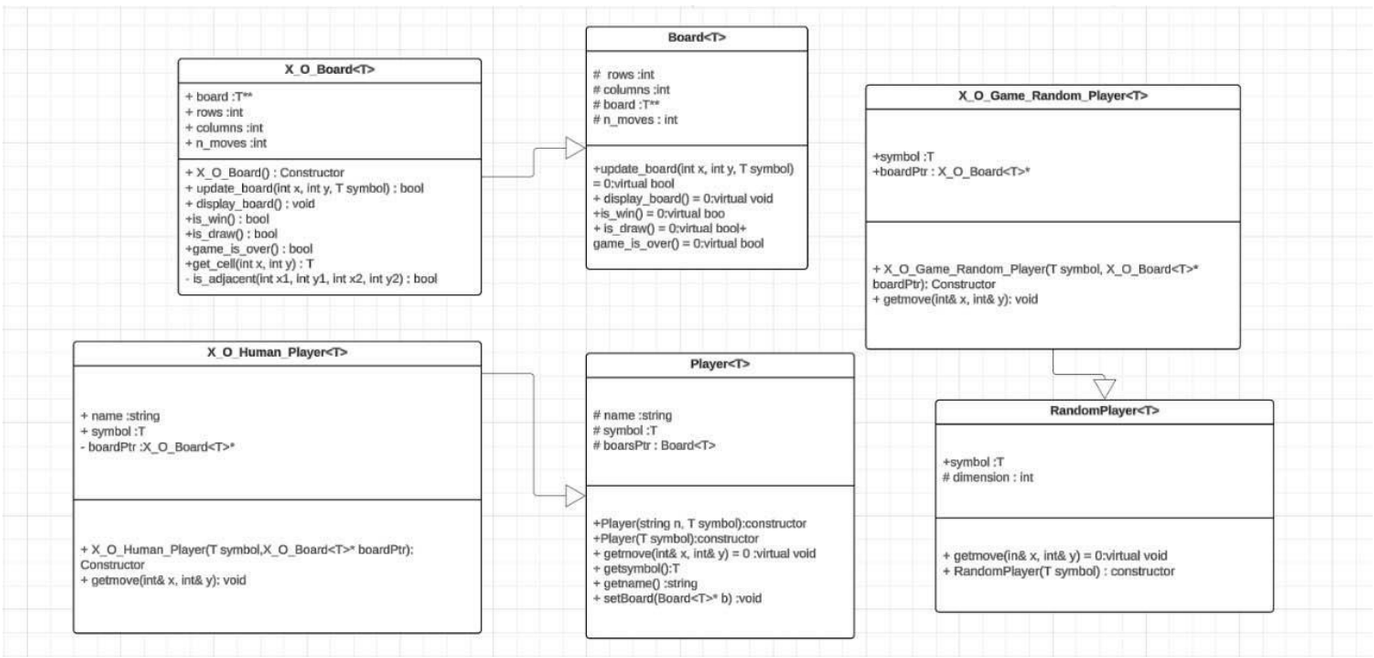
- Implements a basic AI that selects random valid moves.

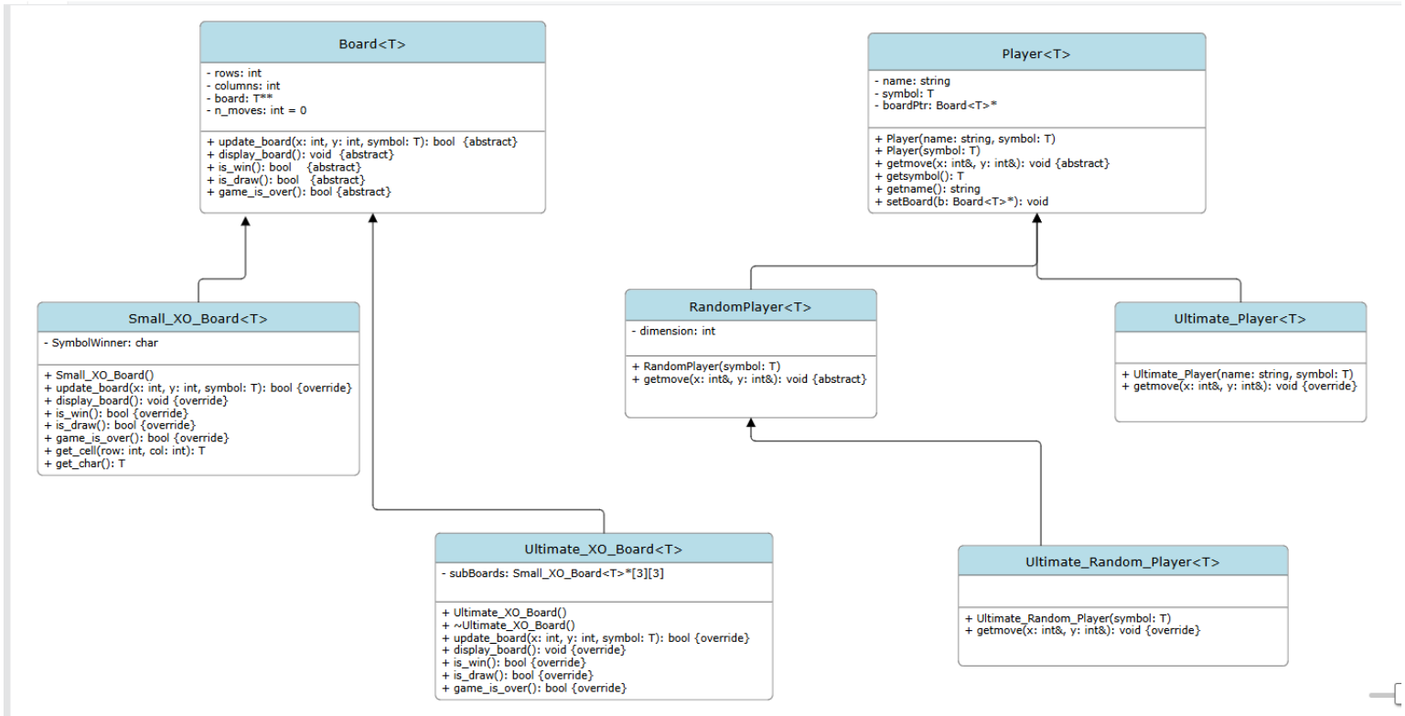
- **Rules:**
  - Ensures the move adheres to adjacency, ownership, and empty cell rules**.**
- **Behavior:**
  - Randomly selects a valid cell and destination, offering a simple yet functional AI experience.

Template Design

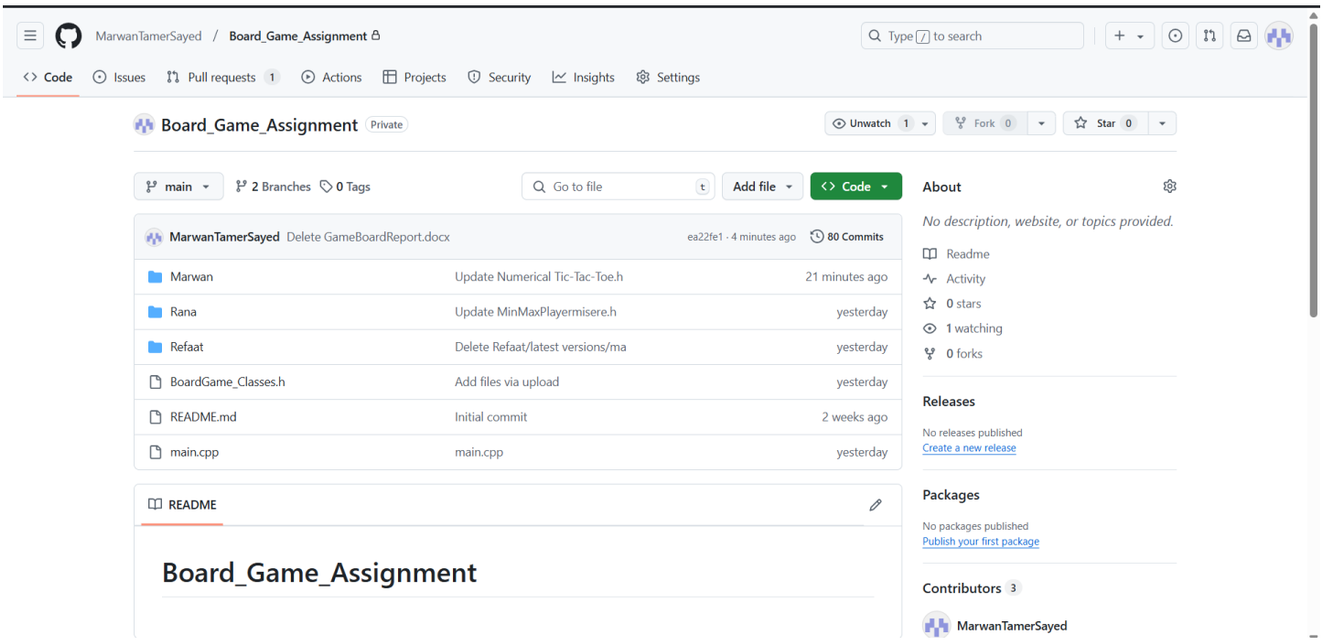
- Generic Implementation:
  - Uses templates (<T>) to adapt the game for various token types (e.g., char, int).

UML:





GitHub:


MarwanTamerSayed / Board_Game_Assignment
Type / to search
Code
Issues
Pull requests 1
Actions
Projects
Security
Insights
Settings
Board_Game_Assignment
Private
Unwatch 1
Fork 0
Star 0
main
2 Branches
0 Tags
Go to file
t
Add file
Code
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
MarwanTamerSayed Delete GameBoardReport.docx
ea22fe1 · 4 minutes ago
80 Commits
Marwan Update Numerical Tic-Tac-Toe.h 21 minutes ago
Rana Update MinMaxPlayermisere.h yesterday
Refaat Delete Refaat/latest versions/ma yesterday
BoardGame_Classes.h Add files via upload yesterday
README.md Initial commit 2 weeks ago
main.cpp main.cpp yesterday
README
Board_Game_Assignment
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 3
MarwanTamerSayed