Faculty of Computers and Artificial Intelligence

Cairo University

CS213: Object Oriented Programming

Under supervision of Dr. Mohamed El-Ramly

Assignment 2

Game Application Report

Name	Ahmed Mustafa El-Shiekh	Nour Hany Salem	Malak Khaled
Emails	am1804067@gmail.com	nourhanysalem2005@gmail.com	malakkhaled0005@gmail.com
ID	20230636	20230447	20231171

➢ Classes Description:

1.Pyramic Tic-Tac-Toe

★ PTTT_Board

- ❖ ***Purpose*:** Represents a pyramid-shaped board for the Pyramid Tic-Tac-Toe game.
- ❖ ***Key Features:***
  - ❑ The board has:
    - ***Row 0:*** 1 cell.
    - ***Row 1:*** 3 cells.
    - ***Row 2:*** 5 cells.
  - ❑ Implements functions for:
    - ***Updating the board:*** Ensures moves are valid and updates the board if the chosen cell is empty.
    - ***Displaying the board:*** Outputs the board using proper formatting to match the pyramid shape.
    - ***Win detection:*** Checks rows, columns, and diagonals for winning patterns.
    - ***Draw detection:*** Game is a draw if 7 or more moves are made without a win.
  - ❑ ***Custom memory allocation:*** Dynamically creates a pyramid structure for the board and deallocates memory in the destructor.
- ❖ ***Inheritance:*** Extends Board<char>.

★ PTTT_Player

- ❖ **Purpose:** Represents a human player in Pyramid Tic-Tac-Toe.
- ❖ **Key Features:**
  - ❑ **Input Handling:**
    - Accepts and validates player input for coordinates.
    - Adjusts input (1-based) to fit the array indexing (0-based).
- ❖ **Inheritance:** Extends Player<char>.

★ PTTT_Random

- **Purpose:** Represents a random AI for the Pyramid Tic-Tac-Toe game.
- **Key Features:**
  - Randomly selects valid moves based on the structure of the pyramid:
    - Row 0: Only column 2 is valid.
    - Row 1: Columns 1, 2, and 3 are valid.
    - Row 2: Any column from 0 to 4 is valid.
- **Inheritance:** Extends RandomPlayer<char>.

★ PTTT_Gameplay

- **Purpose:** Manages the overall Pyramid Tic-Tac-Toe gameplay and user interface.
- **Key Features:**
  - Provides a console-based user interface to:
    - Welcome the player and set up the game.
    - Prompt the player for names and choose the game mode: **Player vs Player** or **Player vs Random**.
  - Handles the gameplay loop using the play() method inherited from Gameplay<char>.
  - Dynamically initializes the pyramid-shaped board and player objects.
- **Inheritance:** Extends Gameplay<char>.

2.Four in Row

★ C4_Board

- ***Purpose:*** Represents the game board for Four-in-a-Row.
- ***Key Features:***
  - Stores the board's state as a 6x7 grid.
  - Tracks the number of moves made and the status of each column (filled rows).
  - Methods to update the board, check for wins, check for draws, and display the board.
  - Includes helper methods to normalize the board for game theory checks and flatten it for AI analysis.
- ***Inheritance:*** Extends Board<char>.

★ C4_Player

- ***Purpose:*** Represents a generic player in the game.
- ***Key Features:***
  - Allows players to input their moves manually.
  - Can be extended for specific player types like AI or random players.
- ***Inheritance:*** Extends Player<char>.

★ C4_ai

- ***Purpose:*** Implements an AI player for the game.
- ***Key Features:***
  - Uses a minimax-like algorithm (max_move) to compute the best move.
  - Considers future moves up to a certain depth (max_depth) and evaluates the board's score recursively.
  - Makes decisions based on game state provided by the C4_Board class.
- ***Inheritance:*** Extends C4_Player.

★ C4_Random

- ***Purpose:*** Represents a player that makes random moves.
- ***Key Features:***
  - Randomly selects a column for the move without considering the game state.
- ***Inheritance:*** Extends RandomPlayer<char>.

Purpose: Manages the overall gameplay and user interface.

- ***Key Features:***
  - Initializes the board and players.
  - Provides an interactive console-based interface to select game modes:
    - Player vs Player
    - Player vs AI
    - Player vs Random
  - Manages the game flow by calling the appropriate methods to take turns and determine outcomes.
- **Inheritance:** Extends Gameplay<char>.

3.FxF Tic-Tac-Toe

★ **FxF_Board**

❖ ***Purpose:***

❖ Implements a **5x5 Tic-Tac-Toe** board for a two-player game. Players compete to win by forming patterns (rows, columns, or diagonals).

❖ ***Key Features:***

❑ **The board has:**

- A 5x5 grid initialized with empty cells (`' - '`).

❑ **Implements core functions:**

- **Updating the board:**

Ensures moves are valid (within bounds and in empty cells) and updates the board with the player's symbol.

- **Displaying the board:**

Displays the board with proper row and column indices for clarity.

- **Win detection:**
  - Checks for winning patterns in rows, columns, and diagonals using predefined kernels.
  - Uses helper functions like `GameTheory::checkWinner()` for efficient pattern matching.
- **Draw detection:**

Declares a draw if all 24 moves are made without a winner.

- **Game Over:**

Combines win and draw checks to determine when the game ends.

❑ **Score tracking:**

Updates scores for both players and determines the winner at game conclusion.

❖ ***Memory Management:***

❑ Dynamically allocates a 2D array for the board.

❑ Cleans up memory when the game ends.

★ **FxF_Player**

❖ ***Purpose:***

Represents a human player who interacts with the game.

❖ **Key Features:**

❑ **Move input:**

Prompts the player to input their move (row and column).

★ **FxF_Random**

❖ ***Purpose:***

Implements a simple AI player that generates random moves.

❖ **Key Features:**

❑ **Random move generation:**

Chooses valid random positions on the board.

4.Word Tic-Tac-Toe

★ WTTT Board

- ***Purpose:*** Represents a 3x3 board for the Word Tic-Tac-Toe game where letters form valid words.
- ***Key Features:***
  - **The board:**
    - 3 rows, 3 columns (3x3 grid).
  - **Implements functions for:**
    - ***Updating the board:*** Ensures moves are valid, updates cells if empty, and tracks moves.
    - ***Displaying the board:*** Outputs the grid with row and column indices.
    - ***Win detection:*** Checks horizontal, vertical, and diagonal patterns for valid words using a dictionary (`dic.txt`).
    - ***Draw detection:*** Declares a draw if all 9 moves are made without a win.
  - ***Custom memory allocation:*** Dynamically creates the 3x3 board and deallocates memory in the destructor.
- ***Inheritance:*** Extends `Board<char>`.

★ WTTT Player

- ***Purpose:*** Represents a player for the Word Tic-Tac-Toe game.
- ***Key Features:***
  - **Player information:**
    - **Name:** Identifies the player.
    - **Symbol:** Represents the letter (A-Z) chosen by the player.
  - **Implements functions for:**
    - ***Getting player move:***
      - Prompts the player to:
        - Enter an alphabet (letter to place).
        - Enter row and column coordinates to place the letter on the board.
      - Converts the input letter to uppercase for consistency.

★ **WTTT Random Player**

- ***Purpose:*** Represents a randomized player for Word Tic-Tac-Toe.
- ***Key Features:***
  - **Implements functions for:**
    - ***Random move generation:***
      - Randomly selects a row and column (3x3 grid).
      - Randomly chooses an alphabet (A-Z) to place.

5.Numerical Tic-Tac-Toe

★ **NTTT_Board**

- ***Purpose:*** Represents the board for the Numerical Tic-Tac-Toe game.
- ***Key Features:***
  - ***Dimensions:*** 3x3 grid initialized with zeros.
  - Tracks the number of moves and updates the board with a given symbol (number).
  - Methods for:
    - ***Updating the board:*** Checks if a move is valid and places the symbol.
    - ***Displaying the board:*** Prints the board to the console.
    - ***Game-over checks:*** Determines if the game is a draw or a win using kernels for horizontal, vertical, and diagonal patterns.
  - Uses the GameTheory::checkWinner utility to identify a winning state based on the sum of numbers in a row (e.g., 15 for NTTT).
- ***Inheritance:*** Extends Board<int>.

★ **NTTT_Player**

- ***Purpose:*** Represents a human player in the Numerical Tic-Tac-Toe game.
- ***Key Features:***
  - ***Input Handling:***
    - Players specify the row and column to place their number.
    - Validates the number to ensure:
      - Odd numbers for Player 1 (symbol 1).
      - Even numbers for Player 2 (symbol 2).
      - Numbers are between 1 and 9 and are not reused.
  - Prompts the user for valid inputs iteratively until a valid move is made.
- ***Inheritance:*** Extends Player<int>.

★ NTTT_Random

- ***Purpose:*** Represents a player that makes random moves in the game.
- ***Key Features:***
  - Randomly selects a row and column for a move.
  - Chooses a random even number from the set {2, 4, 6, 8} to place on the board.
  - Doesn't analyze the game state, purely based on randomness.
- ***Inheritance:*** Extends RandomPlayer<int>.

★ NTTT_Gameplay

- ***Purpose:*** Manages the overall gameplay and user interface for the NTTT game.
- ***Key Features:***
  - Provides an interactive console-based UI for setting up the game:
    - Prompts the user for Player 1's name.
    - Offers two modes: **Player vs Player** and **Player vs Random**.
    - Initializes the NTTT_Board and players accordingly.
  - Handles the game loop using the inherited play() method.
  - Provides feedback to the players during gameplay (e.g., win, draw, or invalid moves).
- **Inheritance:** Extends Gameplay<int>.

6.Misere Tic-Tac-Toe

★ ITTT_Board

❖ **Purpose:** Represents the **3x3 board** for **individual sub-boards** within the Ultimate Tic-Tac-Toe game.
❖ **Key Features:**
❑ **Initialization:**
- Board dimensions: **3x3** initialized with zeros (`0` for empty cells).
- Tracks **number of moves** (`n_moves`) and the **last symbol** placed.

❑ **Methods for:**
- **Updating the board:**
  - Ensures valid moves (within bounds, not occupied).
  - Places a symbol ('X' or 'O') and increments move count.
- **Displaying the board:**
  - Outputs the board with formatted rows and columns.
- **Game-over checks:**
  - **Win detection:**
    - Uses **kernels** for checking horizontal, vertical, and diagonal patterns.
    - Normalizes the board and uses `GameTheory::checkWinner` to identify a win.
  - **Draw detection:**
    - Declares a draw when all 9 cells are filled.

❖ **Dependencies:**
- `GameTheory` utility for checking winning patterns.

★ ITTT_Player

❖ **Purpose:** Handles input for a **human player** in the game.
❖ **Key Features:**
❑ Prompts the player for a move:
- Accepts input in the form `row` and `column`.
- Converts input to **zero-based indices** for board placement.

❖ **Inheritance:** Extends `Player<char>`.

★ **ITTT_Random**

❖ **Purpose:** Represents a player that makes **random moves**.

❖ **Key Features:**

❑ Randomly selects:
- Row (x) and column (y) within the **3x3 grid**.

❑ Ensures moves stay within valid bounds.

❖ **Inheritance:** Extends `RandomPlayer<char>`.

7.Ultimate Tic-Tac-Toe

★ UTTT_Board

- ❖ **Purpose:** Represents the **9x9 grid** divided into **9 sub-boards** for the Ultimate Tic-Tac-Toe game.
- ❖ **Key Features:**
- ❑ **Board Structure:**
  - **Main board:** 3x3 grid where each cell represents a sub-board.
  - **Sub-boards:** Each sub-board is a standard 3x3 Tic-Tac-Toe game.
- ❑ **Methods for:**
  - **Updating the board:**
    - Ensures the move is valid (within the correct sub-board).
    - Updates the corresponding **sub-board** and checks for a win.
    - If a player wins a sub-board, the **main board** is updated.
  - **Displaying the board:**
    - Outputs the full 9x9 grid with formatting to distinguish sub-boards.
  - **Game-over checks:**
    - **Win detection:** Checks rows, columns, and diagonals in the **main board** for a win.
    - **Draw detection:** All sub-boards are full, and no win occurs.
- ❑ **Inheritance:** Extends `Board<char>`.

★ UTTT_Player

- ❖ **Purpose:** Represents a human player for the Ultimate Tic-Tac-Toe game.
- ❖ **Key Features:**
- ❑ **Input Handling:**
  - Prompts the player to specify the **row** and **column** for their move.
  - Ensures moves are valid based on the state of the current sub-board.
- ❑ **Inheritance:** Extends `Player<char>`.

★ *UTTT_Random*

❖ **Purpose:** Represents a player that makes **random moves** in the Ultimate Tic-Tac-Toe game.

❖ **Key Features:**

❑ Randomly selects:

- A **row** and **column** in the appropriate sub-board.
- Ensures moves are valid.

❑ **Inheritance:** Extends `RandomPlayer<char>`.

★ UTTT_Gameplay

❖ **Purpose:** Manages the overall gameplay and user interface for the Ultimate Tic-Tac-Toe game.

❖ **Key Features:**

❑ **Game Setup:**

- Prompts the user for **Player 1's name**.
- Allows two modes:
  - **Player vs Player**
  - **Player vs Random**

❑ **Game Loop:**

- Initializes the `UTTT_Board` and players.
- Handles turns, ensures moves are valid, and provides feedback:
  - **Win detection:** Declares a winner when the main board shows a winning pattern.
  - **Draw detection:** Declares a draw when no valid moves remain.

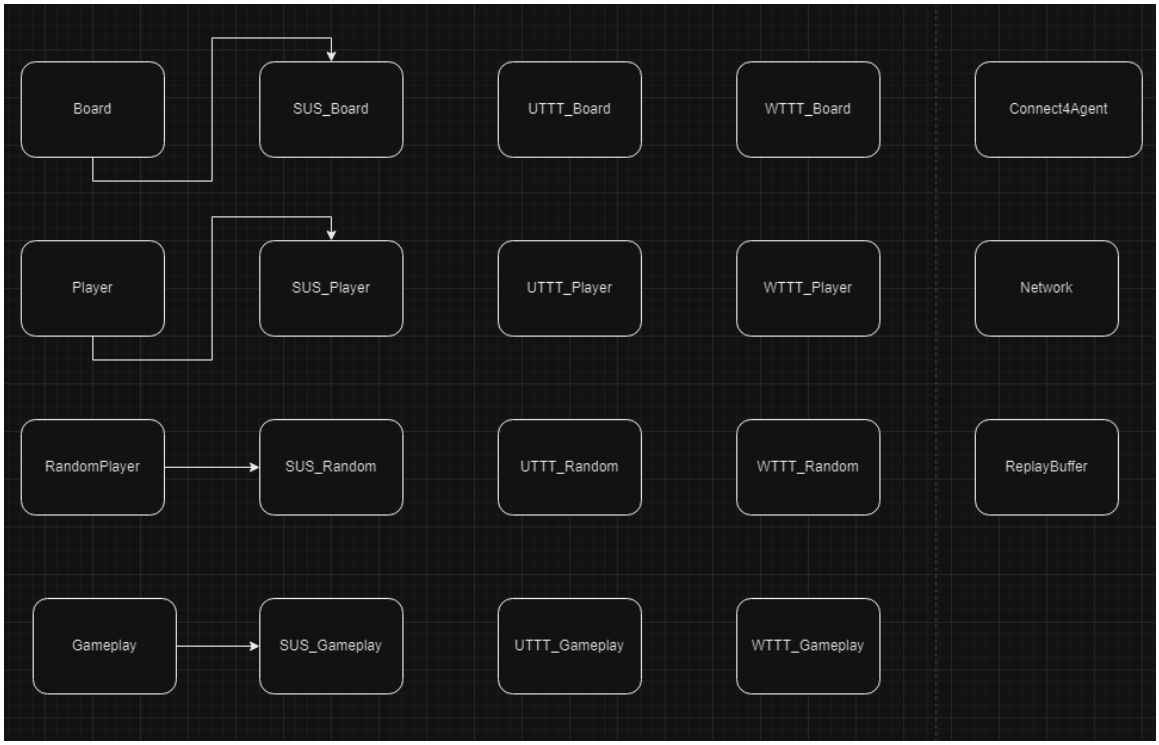
❑ **Inheritance:** Extends `Gameplay<char>`.

8.SUS

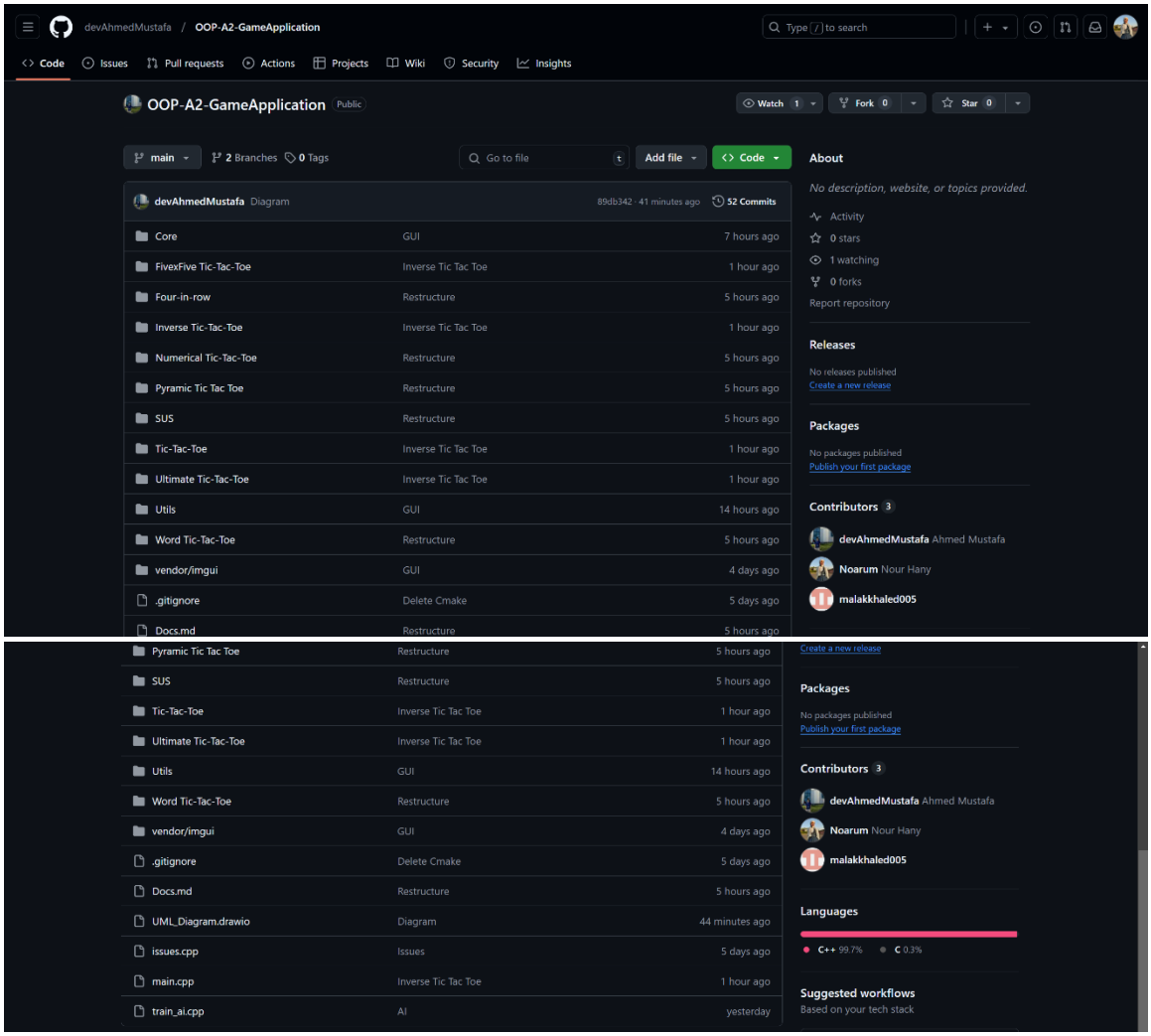
❖ **Key features:**

- ❑ **Dynamic Board Management**: A dynamically allocated board with methods for updating and displaying the game state.
- ❑ **Win and Draw Detection**: Uses kernel-based pattern matching for win conditions and checks for draw scenarios.
- ❑ **Multiplayer and AI Support**: Includes human player (`SUS_Player`) and random AI (`SUS_Random`) implementations.
- ❑ **GUI Integration**: Uses ImGui for interactive button-based gameplay.
- ❑ **Inheritance**:
  - **`SUS_Random`** inherits from `RandomPlayer` to add random behavior and symbol selection.

UML :


Board
SUS_Board
UTTT_Board
WTTT_Board
Connect4Agent
Player
SUS_Player
UTTT_Player
WTTT_Player
Network
RandomPlayer
SUS_Random
UTTT_Random
WTTT_Random
ReplayBuffer
Gameplay
SUS_Gameplay
UTTT_Gameplay
WTTT_Gameplay

Git Hub Repository :



➢ Work Assignment breakdown:

Ahmed Mustafa Elsheikh	Nour Hany Salem	Malak Khaled
▪ Four in Row + AI ▪ Numerical Tic-Tac-Toe	▪ Pyramic Tic-Tac-Toe ▪ Word Tic-Tac-Toe ▪ Report	▪ 5x5 Tic-Tac-Toe ▪ Misere Tic-Tac-Toe
▪ Ultimate Tic-Tac-Toe ▪ SUS		