A logo of a egyptian god

Description automatically generatedA computer and earth with arrows

Description automatically generated with medium confidence

Object Oriented Programming Course (CS213) Under Supervision Eng. Rana Abdelkader

Group: S21 Assignment 3

|  |  |  |
| --- | --- | --- |
| **Name** | **ID** | **Tasks** |
| **Ghassan Tarek Elgendy** | 20220239 | Four-in-a-row Game |
| **Rawan Ehab Ammar** | 20220133 | Pyramic Tic-Tac-Toe |
| **Jana Mohamed Ramadan** | 20220470 | 5 x 5 Tic Tac Toe |

**Pyramic Tic-Tac-Toe:**

1. **Constructor:**

* The constructor initializes an empty 3\*5 array to make the basic dimensions of the pyramid.
* All elements in the array are initialized to the ascii of 0 which is a **blank space.**

1. **Display\_board():**

* Function that prints the content of the array in the relevant parts of the pyramid.
* The pyramic shape is made using the equation 2\*i+1.
* If the content of the elemnt is a blank space the index is printed, else the content of the cell is printed.

1. **Update\_board():**

* Takes the user input and updates the elements of the array if there are no other elements already in that cell and it is a valid cell.
* Increments the number of moves counter to keep track.

1. **Is\_winner():**

* Function to check after every move if there is a winner or not.
* It iterates over each cell if there are three consecutive elements of the same label In any direction , then there is a winner.

1. **Is\_draw():**

* Function to check if the game is a draw
* If 9 moves have been made and there is no winner yet, then it’s a draw.

1. **Game\_is\_over():**

* Returns true when the game is finished

1. **initPyramicTicTacToe():**

* Used to initialize the pyramic Tic-Tac-Toe game in source file.

**Four-in-a-row:**

1. **Constructor ConnectFourBoard:**
   * Initializes a Connect Four board with dimensions 7x7.
   * Populates the board with zeros and fills the bottom row with '1's.
2. **update\_board:**
   * Checks if the move is valid and places the mark on the board if valid.
   * Returns true if the move issuccessful, false otherwise.
3. **display\_board:**
   * Displays the current state of the Connect Four board, including cell coordinates.
4. **is\_winner:**
   * Checks for a winning condition by examining rows, columns, and diagonals.
   * Returns true if there's a winner, false otherwise.
5. **is\_draw:**
   * Determines if the game has ended in a draw.
   * Returns true if the maximum moves (42) are reached without a winner.
6. **game\_is\_over:**
   * Checks if the game is over (all moves are completed or there's a winner).
   * Returns true if the game is finished.
7. **get\_board\_value and set\_board\_value:**
   * Retrieve and set values at specific positions on the board, respectively.

**GameManager Class:**

* **Constructor GameManager(Board\* bPtr, Player\* playerPtr[2]):**
  + Initializes the game manager with a board and two players.
* **run:**
  + Controls the game flow:
    - Displays the initial board state.
    - Loops through player moves, checks for a winner or draw, and ends the game accordingly.

**AiPlayer Class:**

* **Constructor AiPlayer(char symbol):**
  + Initializes an AI player with a given symbol**.**
* **get\_move :**
  + Implements AI logic to make moves based on the current board state.
  + Attempts to win if possible, otherwise blocks the opponent or makes a random move.
* **get\_board :**
  + Retrieves the board pointer for AI decision-making.

**5 x 5 Tic Tac Toe:**

**Code quality review according to** [**Code Review Checklist**](https://www.codereviewchecklist.com/)

**Pyramic XO:**

**Requirements**

Met Requirements: The code appears to fulfill the basic requirements of implementing PyramicTicTacToe.

Correct Formatting: Code is correctly formatted.

Unnecessary Whitespace: Whitespace could be further reduced in some areas.

**Best Practices**

Single Responsibility: Code seems to follow this principle, separating concerns into different classes.

Error Handling: Basic error handling exists but might need improvement for more comprehensive error prevention.

Logging Errors/Warnings: No explicit logging for errors or warnings in the provided code.

Magic Values: Constants could replace some hardcoded values.

Comments: Comments are present but might be insufficient for complex logic understanding.

Nesting: Generally minimal nesting.

**Maintainability**

Readability: Code readability is moderate, but more comments could enhance it.

Code Duplication: Some sections might benefit from refactoring to remove duplication, but they are all necessary for functionality.

Method/Class Length: Functions are all manageable in size.

**Performance**

Performance Acceptance: Basic operations seem fine.

**Architecture**

Separation of Concerns: Different aspects are separated into classes, following a basic separation of concerns.