Cairo University, Faculty of Artificial Intelligence

FACULTY OF COMPUTERS AND ARTIFICIAL INTELLIGENCE, CAIRO UNIVERSITY

CS213: Programming II
Year 2025-2026
First Semester

Assignment 3 – Version 5.0

Course Instructors:

Dr. Mohammad El-Ramly

Revision History

Version 1.0	By Dr Mohammed El-Ramly	26 Nov. 2022	Main Doc
Version 2.0	By Dr Mohammed El-Ramly	26 Nov. 2023	Revised
Version 3.0, 4.0	By Dr Mohammed El-Ramly	26 Nov. 2024	Added new games
Version 5.0	Dr El-Ramly, Gamal, Fatema 2027	6 Nov. 2025	New games & library

Table of Contents

Objectives

This assignment aims to (1) teach OOP concepts in C++, (2) using OOP to build systems with intermediate complexity, (3) Templates and STL, (4) Recursion and Backtracking, (5) Code Reuse.

Instructions

1. Part1: **Deadline** is **13 Nov** 2025. (Task 1.1 - **One** group game and menu)
2. Part2: **Deadline** is **26 Nov** 2025. (Task 1.2 - **Four** individual games + **Two** group games and menu)
3. Part3: **Deadline** is **6 Dec** 2025. (Tasks 1, 2, 3 - The full gaming application and report and bonus)
4. **Weight** is **9** marks + 2 **bonus** marks.
5. Students will form **teams of FOUR students from the same lab/section.**
6. **All team together will implement game 9 and 10 and 11 (and 12 to 13 if they do bonus)**
7. **Smallest ID will individually implement games 1,2. Next one will do 3,4. Next one will do 5,6 and largest id will do 7, 8.**

Academic Integrity Policy

All submitted work must be your **own original effort**. Copying solutions from friends, books, or any internet source, or using AI tools to complete the assignment, will unfortunately result in you failing the course. If you have questions or need clarification, please consult me directly; do not copy solutions from others or external sources.

(تسليم حلول منقولة من أى مصدر يؤدى إلى الرسوب فى هذا المقرر، لا تغش الحل أو تنقله من أى مصدر و اسألنى فى أى شئ لا تفهمه، لكن لا تنقل الحلول من الذكاء الاصطناعي، المواقع المختلفة، زملائك، أو من أى مكان)

Tasks Breakdown

Task 0 (0 marks) - Individual / Team

1. Review OOP / STL / Templates / Recursion / Backtracking.
2. Review **all code examples** given in lecture and in classroom page.
3. Create a **private (NOT public) GitHub** repo for project, invite team members to **use it for development.**
4. Study the given Board Game framework and associated html documentation generated by doxygen.

Task 1 (7 marks) – Individual / Group - OOP Design and Development

Board Game Framework Overview

This assignment provides a generic, object-oriented framework designed for the rapid development of board games. It showcases the power of Object-Oriented Programming (OOP) principles such as modularity and reusability, enabling you to build new games efficiently without starting from scratch. You easily create new games by inheriting from the framework's core classes and overriding some methods. You only need to add the logic specific to the new game. You are not allowed to change the given classes, but you can extend them (inherit from them) **except GameManager**, to create game specific classes like Move, Player, UI and Board. Leave GameManager as is so it is able to run any kind of game.

Core Framework Classes

The framework is comprised of the following generic classes, all contained within the **BoardGame_Classes.h** header file (due to template implementation requirements):

Board<T> is an abstract class . It represents the main game board for any board game. It stores the grid, tracks moves, and defines virtual methods for updating, displaying, and checking game state (win/draw/loss). You need to inherit from it to create specific boards for specific games.

Move<T> is a concrete class. It models a single move in the game. Stores the move's position (x, y) and the player's symbol placed on the board. You can inherit it to create special types of moves for special games.

Player<T> is a concrete class. It defines a player (human or computer) in the game. It stores name, symbol, type, and a pointer to the game board. You can inherit from it to create special types of players for special games. Or you can use it as it is if enough for the game you are developing.

UI<T> is an abstract class. It defines a generic UI for the the game. You need to inherit from it to create a specific UI for your game. It handles user interaction and display. It handles input/output, displays the board, and gathers moves from players.

GameManager<T> is a concrete. Do not inherit from it. It can play any game if you pass to it a board, players and UI of the desired game. It controls the overall game flow between two players. It coordinates turns, updates the board, and determines win/draw/end conditions.

With **BoardGame_Classes.h**, you have a dictionary file **dic.txt** that you will need in some games and also html documentation of the Framework. There is also an example implementation and demo for an XO game as an example.

See the class diagram to better understand the relations between the classes.

Then after that, there is the code of **`run()`** method in GameManager to show how a game works.

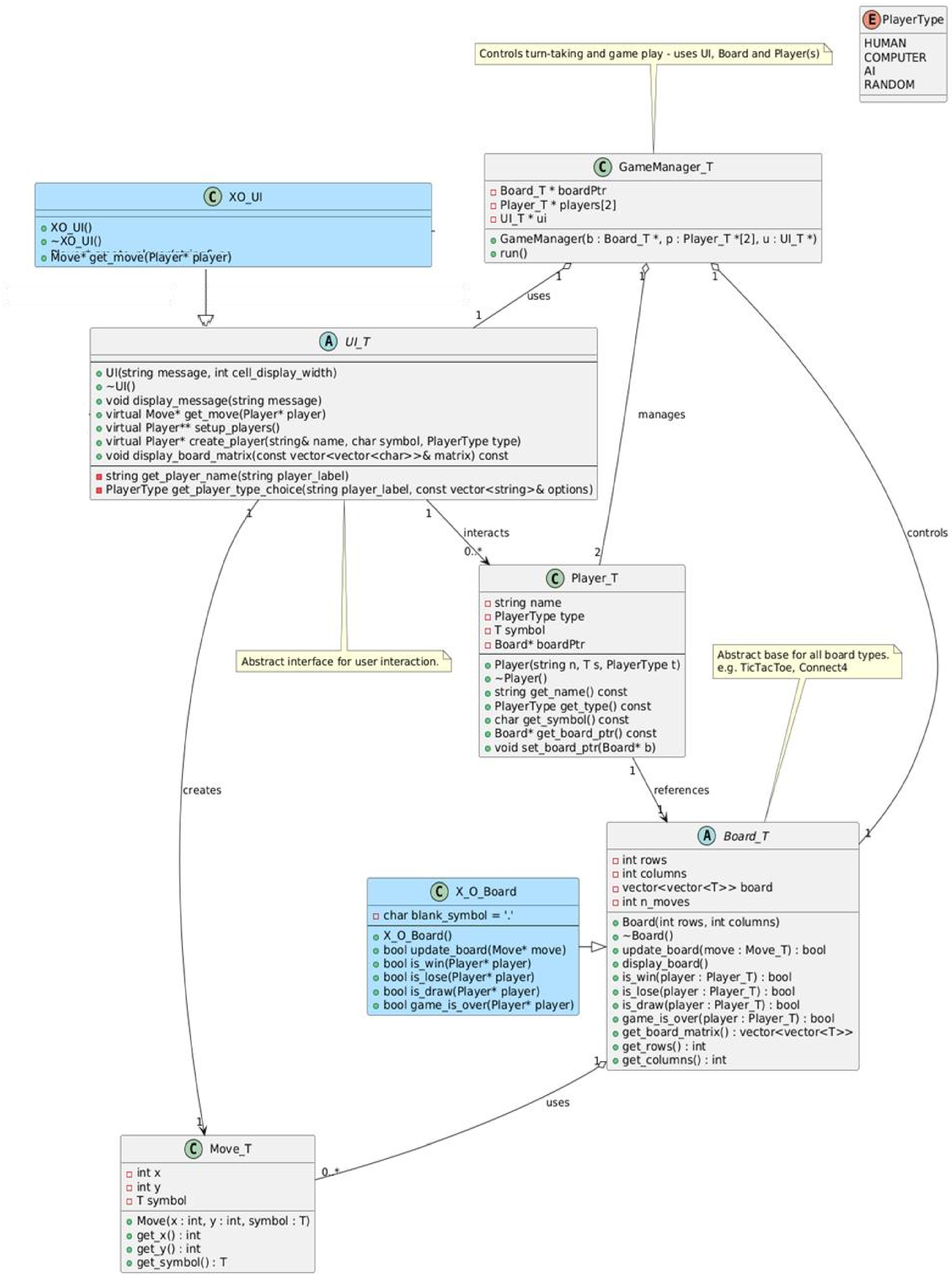
Example X-O Game Implementation

To illustrate the framework's usage, an example X-O game implementation is provided, consisting of these specific classes:

1. **X_O_Board**: Implements the game board and specific rules for Tic-Tac-Toe.
2. **X_O_UI**: Represents a Tic-Tac-Toe UI handling user input and output.

Additionally, the game allows playing against the computer but implements a random player not with AI.

Board Game Core Classes Overview (sanitized for PlantUML)


E PlayerType
HUMAN
COMPUTER
AI
RANDOM
Controls turn-taking and game play - uses UI, Board and Player(s)
C GameManager_T
Board_T * boardPtr
Player_T * players[2]
UI_T * ui
GameManager(b : Board_T *, p : Player_T *[2], u : UI_T *)
run()
C XO_UI
XO_UI()
~XO_UI()
Move* get_move(Player* player)
uses
1
1
manages
1
2
controls
1
1
A UI_T
UI(string message, int cell_display_width)
~UI()
void display_message(string message)
virtual Move* get_move(Player* player)
virtual Player** setup_players()
virtual Player* create_player(string& name, char symbol, PlayerType type)
void display_board_matrix(const vector<vector<char>>& matrix) const
string get_player_name(string player_label)
PlayerType get_player_type_choice(string player_label, const vector<string>& options)
interacts
1
0..*
Abstract interface for user interaction.
C Player_T
string name
PlayerType type
T symbol
Board* boardPtr
Player(string n, T s, PlayerType t)
~Player()
string get_name() const
PlayerType get_type() const
char get_symbol() const
Board* get_board_ptr() const
void set_board_ptr(Board* b)
creates
1
1
references
1
1
Abstract base for all board types.
e.g. TicTacToe, Connect4
C X_O_Board
char blank_symbol = '.'
X_O_Board()
bool update_board(Move* move)
bool is_win(Player* player)
bool is_lose(Player* player)
bool is_draw(Player* player)
bool game_is_over(Player* player)
A Board_T
int rows
int columns
vector<vector<T>> board
int n_moves
Board(int rows, int columns)
~Board()
update_board(move : Move_T) : bool
display_board()
is_win(player : Player_T) : bool
is_lose(player : Player_T) : bool
is_draw(player : Player_T) : bool
game_is_over(player : Player_T) : bool
get_board_matrix() : vector<vector<T>>
get_rows() : int
get_columns() : int
uses
1
0..*
C Move_T
int x
int y
T symbol
Move(x : int, y : int, symbol : T)
get_x() : int
get_y() : int
get_symbol() : T

```cpp
template <typename T>
void GameManager<T>::run() {
     // Display the current board at start of the game
     ui->display_board_matrix(boardPtr->get_board_matrix());

    // Give turn to first player
    Player<T>* currentPlayer = players[0];

     // Loop while game is not finished
     while (!boardPtr->game_is_over(currentPlayer)) {
         for (int i : {0, 1}) {                          // for each player
             currentPlayer = players[i];                 // give turn
             Move<T>* move = ui->get_move(players[i]);   // get / generate move

             while ((boardPtr != nullptr) && (!boardPtr->update_board(move)))// update board
                 move = ui->get_move(players[i]);                    // get move done

             ui->display_board_matrix(boardPtr->get_board_matrix());    // display board

             if (boardPtr->is_win(players[i])) {                     // if winner
                 ui->display_message(players[i]->get_name() + " wins!"); // delcare it
                 return;                                             // end game
             }
             if (boardPtr->is_lose(players[i])) {                    // if lost
                 ui->display_message(players[1 - i]->get_name() + " wins!"s);// other is winner
                 return;                                             // end game
             }
             if (boardPtr->is_draw(players[i])) {                    // if draw
                 ui->display_message(string("Draw!"));               // announce it
                 return;                                             // end game
             }
         }
     }
}
```

Task 2 (2 marks) - Group - Doxygen, Integration and Code Quality

This task combines the **integration** of your developed games into a **single application** and generating **code documentation for it**. It also ensures **code quality.**

- **Application Integration**: (Worth 1 mark)
    - The team must integrate all games into one cohesive application.
    - This application should present a menu of available games to the user.
    - Upon selecting a game, it must launch, allowing the user to play against another human player or a computer player (random or smart)

- **Documentation and Code Review**: (Worth **1** mark)
  - **Documentation:** Generate html pages for your code API using Doxygen. You can use AI to generate the doxygen comments for you and then use Doxygen tool to generate web pages. Doxygen download
  - **Code Review Purpose**: Code review is an essential practice in software development for ensuring code quality, enhancing readability, and minimizing bugs.
  - **Process**: Each team member is required to review the code written by another team member.
  - **Reporting**: Write your findings in the project report, detailing any errors or quality issues identified.
  - **Resources for Code Review:**
    - To deepen your understanding of code reviews and available tools, explore:
    - 12 Best Code Review Tools for Developers
    - Awesome Code Review
    - When reviewing the code, utilize a checklist such as this one (or a similar alternative):
      - Code Review Checklist
    - In your report, clearly state any violations found, specifying the file name and line number(s) where they occur.

Tasks 3 (Bonus 3 marks) – Individual / Group - AI Players

Individual - Two marks. For your own **two individual games**, develop a smart AI computer player that uses some smart AI algorithm like backtracking, min-max or others to decide a good move given the current board status instead of playing randomly. The goal is to play against human in order to win.

Read about min-max and backtracking algorithms and choose a suitable algorithm or try a suitable heuristic algorithm that chooses the best move for the computer. Try several game plays to ensure your algorithm works well and makes smart decisions.

Notice that the current X-O game demo has only a random computer player that makes random moves.

Group - One mark. Team can develop games **12, 13 as bonus** as a group together to earn a earn 1 mark.

Team Project & Game Assignments

Collaboration Guidelines

Members are encouraged to help one another, but each student is ultimately responsible for completing their individually assigned games. Collaboration is key for integrating all games into the final app.

Constraints:

Extending the Framework (Open/Closed Principle). You are **not permitted to modify the existing framework code**. Instead, you must adhere to the Open/Closed Principle: extend functionality by creating **new classes that inherit from the base classes**.

For each new game, you will need to:

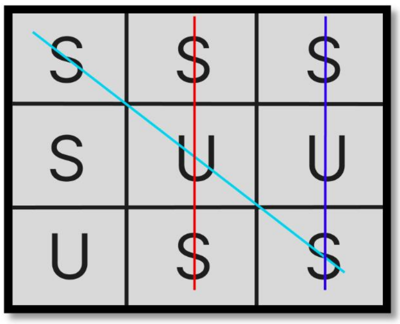
- Inherit from the **Board** class to define its unique rules and board logic.
- Potentially inherit from the **Player** class to create a specialized player (if needed).
- Develop a **random computer player** for that specific game, ensuring it makes valid random moves.
- Refer to the provided X-O game example for guidance on this inheritance approach.

Game Descriptions

Game descriptions are taken from: Tic-Tac-Toe Variations

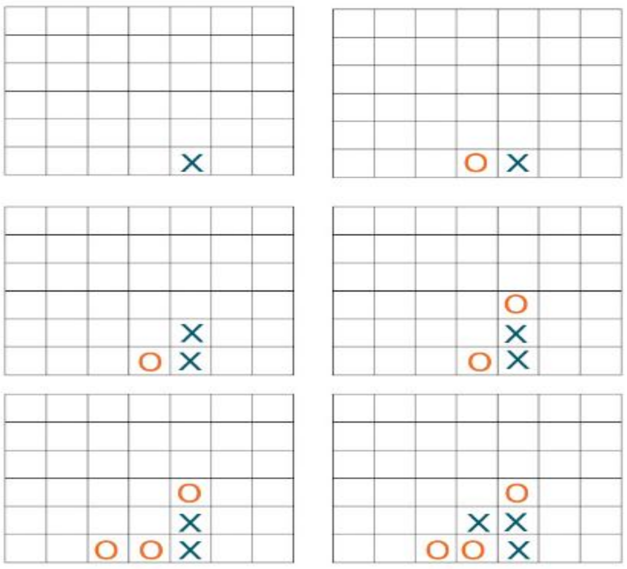
1. SUS

- ✓ **Introduction:** A game on a 3x3 grid where the objective is to form the sequence "S-U-S".
- ✓ **Board:** A 3x3 grid.
- ✓ **Rules:** Players take turns placing either an "S" or a "U" in an empty square. A player must use the same letter each turn. A player scores a point for each "S-U-S" sequence they create.
- ✓ **Winning Condition:** The game ends when the board is full. The player who has created the most "S-U-S" sequences wins.



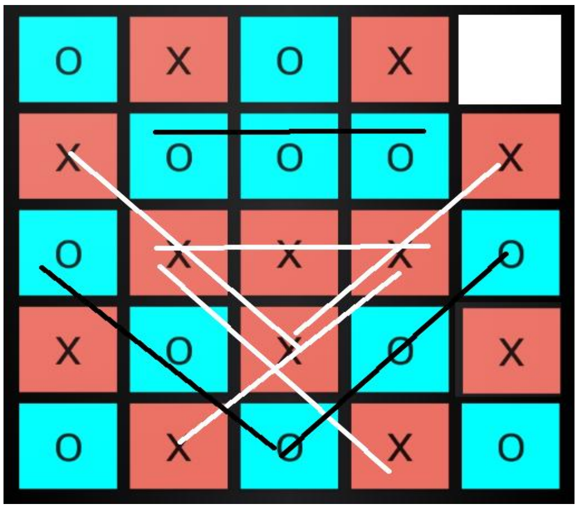
2. Four-in-a-row

- ✓ **Introduction:** A 2D version of the classic game Connect Four where players mark a grid instead of dropping counters.
- ✓ **Board:** A 6 x 7 grid (six rows, seven columns).
- ✓ **Rules:** The first player places an 'X' in the bottom-most square of any column. Subsequent players take turns placing their mark in the lowest available square of any column.
- ✓ **Winning Condition:** The first player to get four of their marks in a row horizontally, vertically, or diagonally wins.



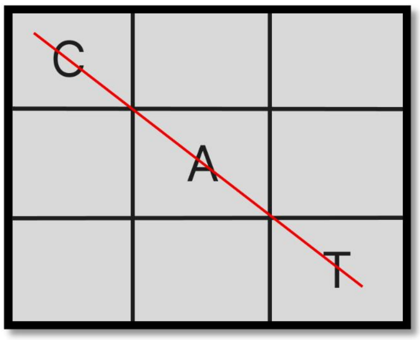
3. 5 x 5 Tic Tac Toe

- ✓ **Introduction:** A variant played on a larger 5x5 grid where the goal is to have the most three-in-a-row sequences.
- ✓ **Board:** A 5x5 grid.
- ✓ **Rules:** Players take turns placing their mark in an empty square until only one square remains empty. This results in a total of 24 moves (12 for each player).



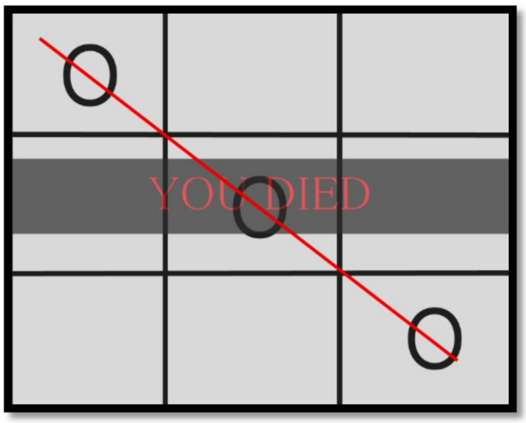
4. Word Tic-tac-toe

- ✓ **Introduction:** A version where players place letters on the board to form valid three-letter words.
- ✓ **Board:** A 3x3 grid.
- ✓ **Rules:** Players take turns placing one letter on the board, attempting to form a valid word. Players can build upon letters already on the board.
- ✓ **Winning Condition:** The first player to form a valid three-letter word horizontally, vertically, or diagonally wins. The game is a draw if the board is filled without a valid word being formed.
- ✓ **Additional Details:** A file named `dic.txt` containing all valid 3-letter words will be provided.



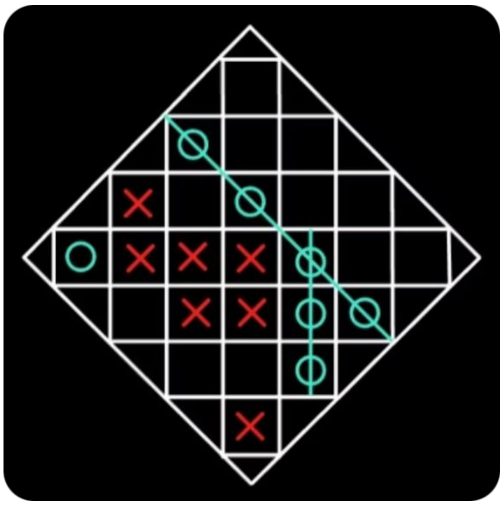
5. Misère Tic Tac Toe

- ✓ **Introduction:** Also known as Inverse Tic-Tac-Toe, the objective is to force your opponent to get three-in-a-row.
- ✓ **Board:** A standard 3x3 grid.
- ✓ **Rules:** The goal is to avoid placing three of your marks in a row, column, or diagonal.
- ✓ **Winning Condition:** A player loses if they complete a line of three of their own marks. If the board is filled with no player achieving three-in-a-row, the game is a draw.



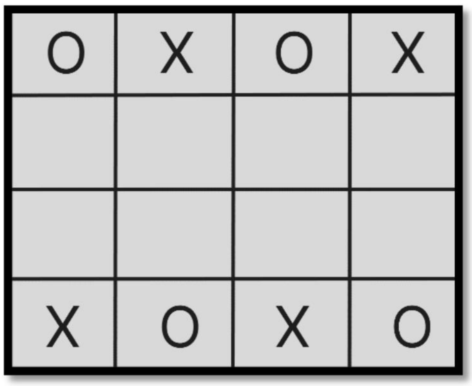
6. Diamond Tic-Tac-Toe

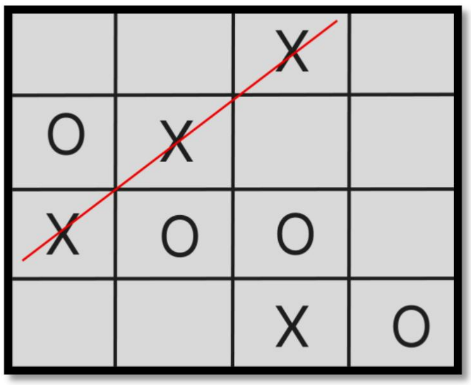
- ✓ **Introduction:** A game on a diamond-shaped grid where a player must complete two lines of marks simultaneously to win.
- ✓ **Board:** A 5x5 grid arranged in a diamond shape.
- ✓ **Rules:** Players take turns placing their 'X' or 'O' in empty cells.
- ✓ **Winning Condition:** A player wins by simultaneously completing a line of three marks and a line of four marks. The two lines must be in different directions (e.g., one horizontal, one diagonal) but can share one common mark.



7. 4 x 4 Tic-Tac-Toe

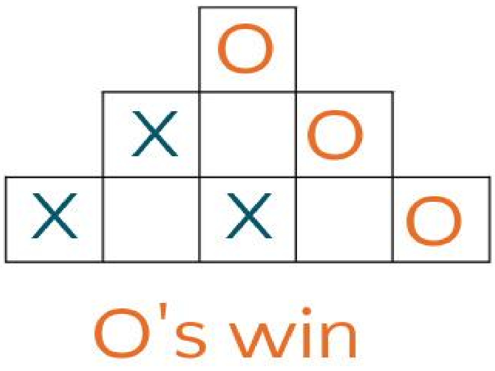
- ✓ **Introduction:** An extended version on a larger board where players move their tokens to create a three-in-a-row alignment.
- ✓ **Board:** A 4x4 grid. Each player starts with four tokens placed in specific starting positions.
- ✓ **Rules:** Players take turns moving one of their tokens to an adjacent (horizontal or vertical) empty square. Tokens cannot move diagonally or jump over other tokens.
- ✓ **Winning Condition:** The first player to align three of their tokens in a row, column, or diagonal wins.

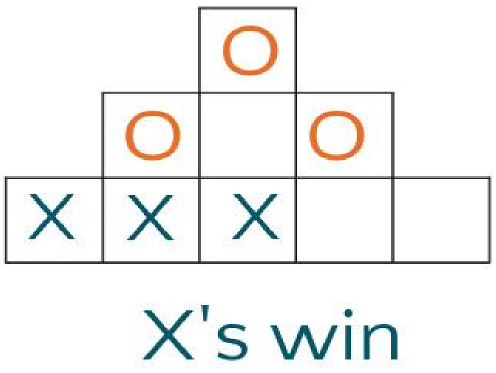




8. Pyramid Tic-Tac-Toe

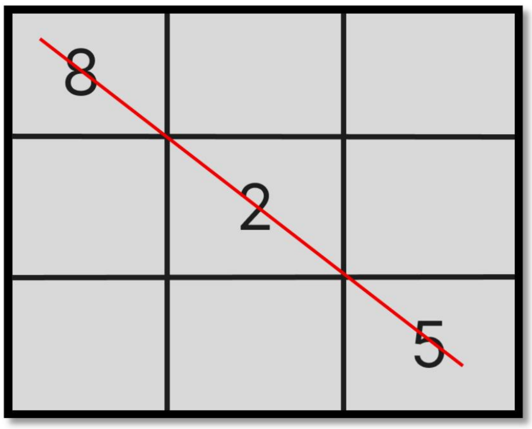
- ✓ **Introduction:** A Tic-Tac-Toe variant played on a pyramid-shaped game board.
- ✓ **Board:** A pyramid structure with a base of five squares, a middle row of three, and one square at the top.
- ✓ **Rules:** Players take turns placing their 'X' or 'O' marks in empty squares.
- ✓ **Winning Condition:** The first player to align three of their marks horizontally, vertically, or diagonally wins.





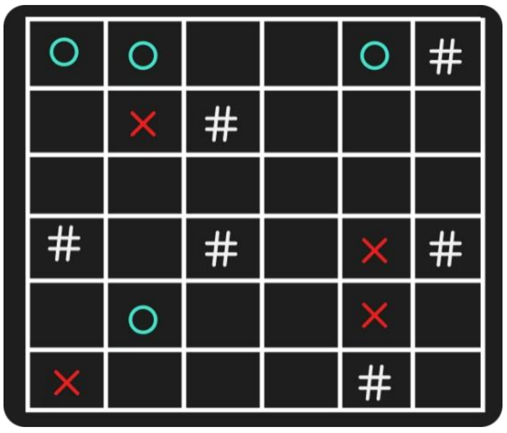
9. Numerical Tic-Tac-Toe (Group)

- ✓ **Introduction:** A mathematical variant where players use numbers with the objective of making a line sum to 15.
- ✓ **Board:** A 3x3 grid.
- ✓ **Rules:** Player 1 uses odd numbers (1, 3, 5, 7, 9) and Player 2 uses even numbers (2, 4, 6, 8). Players alternate placing one number in an empty cell, and each number can only be used once.
- ✓ **Winning Condition:** A player wins by placing three numbers in a row, column, or diagonal that add up to exactly 15. If all cells are filled and no line sums to 15, the game is a draw.



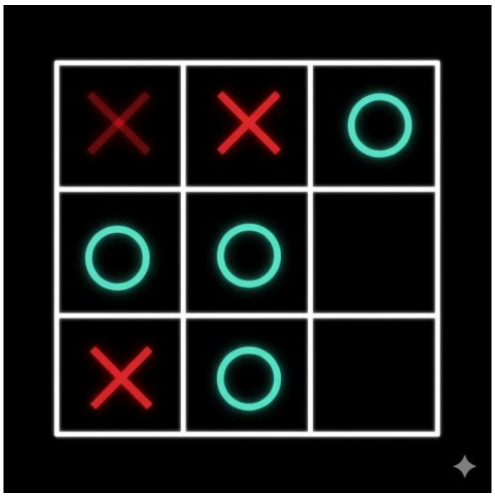
10. Obstacles Tic-Tac-Toe (Group)

- ✓ **Introduction:** A dynamic game where obstacle cells are randomly added to the board, blocking squares.
- ✓ **Board:** A 6x6 grid.
- ✓ **Rules:** After every round (one turn for each player), two new obstacle cells are randomly added to the board. These cells cannot be used by either player.
- ✓ **Winning Condition:** The first player to align four of their marks in a row (horizontally, vertically, or diagonally) wins. The game is a draw if the board fills without a winner.



11. Infinity Tic-Tac-Toe (Group)

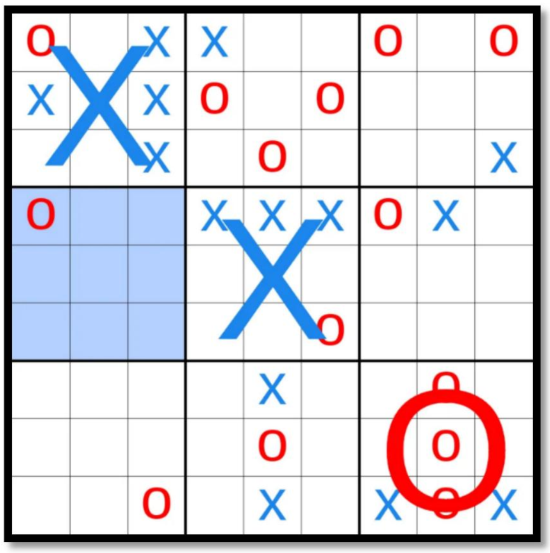
- ✓ **Introduction:** A game on a 3x3 grid where the oldest moves are periodically removed from the board.
- ✓ **Board:** A 3x3 grid.
- ✓ **Rules:** After every three moves, the oldest mark on the board disappears.
- ✓ **Winning Condition:** The first player to align three marks in a row before any of those marks vanish wins the game.



---

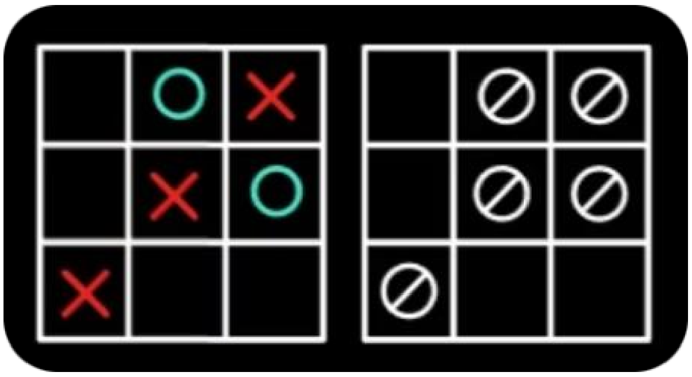
12. Ultimate Tic Tac Toe (Group - Bonus)

- ✓ **Introduction:** A meta-game consisting of a large 3x3 grid where each cell contains a smaller, standard 3x3 Tic-Tac-Toe board.
- ✓ **Board:** A 3x3 main grid, with each of its nine cells containing its own 3x3 sub-grid.
- ✓ **Rules:** Player 1 starts on any of the nine smaller boards. The winner of a smaller board claims that corresponding cell on the main board with their symbol.
- ✓ **Winning Condition:** The first player to win three smaller boards in a row (horizontally, vertically, or diagonally) on the main 3x3 grid wins the game.



13. Memory Tic-Tac-Toe (Group - Bonus)

- ✓ **Introduction:** A variant where marks are hidden after being placed, testing the players' memory.
- ✓ **Board:** A 3x3 grid.
- ✓ **Rules:** When a player places a mark, it is immediately hidden from view. Players must remember the positions of all marks to plan their strategy.
- ✓ **Winning Condition:** The first player to align three of their hidden marks in a row wins. The game is a draw if the board fills without a winner.



What to deliver

1- **Written original non-cheated code in standard C++ not using third-party libraries** representing the whole gaming app that has your team games**. Deliver one integrated app.**
2- Name your file **A3_Sxx_PartX_TAFirstName_YourIDs.zip** (Sxx is your section, PartX is Part1 or Part2 or Part3 depending on the delivery date)
3- **NO EXE. Do not upload exe or object files.**
4- Code should be well organized and commented.
5- Directory with **html doxygen** pages generated for your code
6- **A pdf** report, including:
   a. Cover page with names, emails, ID, logo, course name, prof name, section
   b. A description of the classes you created for each game & **how u used 4 "A PIE" OOP principles**.
   c. A work breakdown table saying **who did what. Specify who did which bonus.**
   d. A report of the **quality of the code** of another member, how u did review, any errors you found.
   e. Screen shots of some of your games.
   f. Image of a **private** project in **GitHub** to collaborate on code.
   g. Explain if you did AI players or bonus games
   h. A link to a **public video to demo some of your games.**
7- Code of the game app organized in files in one directory.
8- **All team members must understand the details of all programs** and be able to explain it.

Marking Criteria

1. 4 For a working original **individual game** that play correctly = 2 x 2. (individual)
2. 3 For correctly working original **group games.** 1 x 3 (group)
3. 1 For **code reviews of all codes** and **documentation** (group)
4. 1 For **integrating** the games together in one program with a menu (group)
5. -1 For not using GitHub. (group)
6. -1 For poor good quality report or no video. (group)

7. 2 For a working smart **AI player** for your 2 games with a suitable algorithm (individual)
8. 1 For correctly working original **bonus group games.** (group)

9. -9 For cheating any part of the code, even 5 lines or generated code.