CAIRO UNIVERSITY
FACULTY OF COMPUTERS AND
ARTIFICIAL INTELLIGENCE

CS213 - Programming II
Object Oriented Programming
2023 / 2024 - First Semester
Teaching Assistant : Dr. Mohammad El-Ramly

Assignment 3

Program Purpose : OOP Design and Development a X-O Games (Pyramid Tic-Tac-Toe / Four-in-a-row / 5 x 5 Tic Tac Toe)

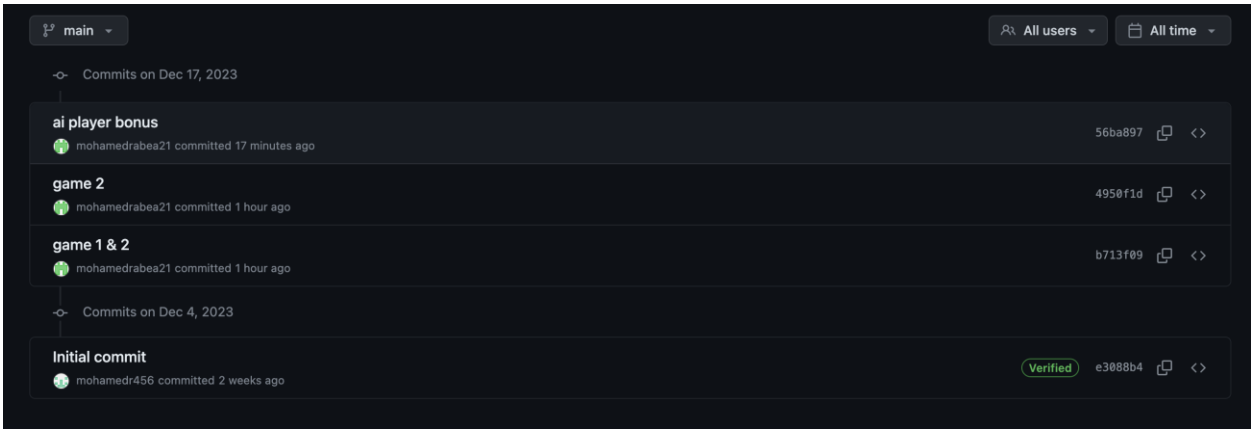
Author Name	ID	GROUP & Section	Mail
Dalia Adel Mohamed	**20220516**	**B-S8**	**daliaadel262@gmail.com**
Mohamed Rabea Mohamed	**20220424**	**B-S8**	**mhmdrby769@gmail.com**

Tasks Table

Tasks / Names	Dalia Adel	Mohamed Rabea
Task 1	Problem sheet 2 (1 – 2 – 5)	Problem sheet 2 (3 – 4 – 6)
Task 2 & 3	Game 3 & integrated app	Game 1 & 2
Part 3	GUI & PVS & Github	AI player & PVS & Github

GitHub screenshots

Mohamed Rabea



Dalia Adel

UML Class Diagram

Code explanation

Game 1

Game 1 is a Pyramid Tic-Tac-Toe :

1. **Abstract Board Class:**
   - **Board** is an abstract class representing a game board. It defines virtual functions for updating the board, checking for a winner, checking for a draw, displaying the board, and determining if the game is over.
2. **Concrete X_O_Board Class:**
   - **X_O_Board** is a concrete class that inherits from the **Board** class. It implements the functions defined in the **Board** class for the specific case of a 3x5 Tic-Tac-Toe pyramid.
3. **Player Class:**
   - The **Player** class represents a player in the game. It has a name and a symbol ('X' or 'O'). There are two constructors—one for a computer player and one for a human player.
4. **RandomPlayer Class:**
   - **RandomPlayer** is a subclass of **Player** representing a computer player that makes random moves within the board boundaries.
5. **GameManager Class:**
   - **GameManager** class manages the game. It has a pointer to a **Board** and an array of two **Player** pointers. The **run** method executes the game loop, taking player moves, updating the board, and checking for a winner or draw.
6. **Initialization (main function):**
   - The main function initializes two players based on user input—either a human player or a random computer player. It then creates an instance of the **GameManager** class with the chosen board type and players and starts the game.
7. **Board Initialization (X_O_Board constructor):**
   - The **X_O_Board** constructor initializes the game board with spaces, forming a pyramid structure. It also sets the number of rows, columns, and total moves to 0.
8. **Update Board (X_O_Board::update_board):**
   - The **update_board** function checks if a move is valid and updates the board if the move is within the board boundaries and the selected cell is empty.
9. **Display Board (X_O_Board::display_board):**
   - The **display_board** function prints the current state of the board, showing the positions of 'X' and 'O' symbols.
10. **Game Loop (GameManager::run):**

- The **run** method manages the main game loop, where players take turns making moves. It checks for a winner or draw after each move and terminates the game when there is a winner or the board is full. The game outcome is then displayed.

Game 2

Game 2 is a *Four-in-a-row :*

1. **Class Hierarchy:**
   - The code defines a hierarchy of classes for a Four-in-a-Row X-O game.
   - Classes include **Board**, **X_O_Board**, **Player**, **FourInARowPlayer**, **RandomPlayer**, **RandomFourInARowPlayer**, **GameManager**.
2. **Board Class:**
   - The abstract base class **Board** provides an interface for game boards with pure virtual functions for updating the board, checking for a winner, a draw, and displaying the board.
3. **X_O_Board Class:**
   - This class represents the specific implementation of the game board for a traditional Tic-Tac-Toe (3x3) game.
4. **Player Class:**
   - Represents a player with a name and a symbol (X or O).
   - Virtual function **get_move** is responsible for getting the player's move.
5. **FourInARowPlayer Class:**
   - Derived from **Player**, this class is specific to the Four-in-a-Row game.
   - Overrides the **get_move** function to take input for the column in this version of the game.
6. **RandomPlayer Class:**
   - Represents a computer player that makes random moves.
   - Overrides the **get_move** function to generate random moves.
7. **GameManager Class:**
   - Manages the flow of the game, including initializing the board and players, and running the game loop.
8. **FourInARowBoard Class:**
   - Represents the game board for the Four-in-a-Row game (7x6).
   - Implements functions to update the board, display it, check for a winner, a draw, and if the game is over.
9. **RandomFourInARowPlayer Class:**
   - Represents a computer player specific to the Four-in-a-Row game that makes random moves.
10. **Main Function:**
    - In the **main** function, it creates instances of players based on user input.
    - It initializes the game manager with a Four-in-a-Row board and the players and runs the game.

Game 3

Game 3 is a 5x5 *tic tac toe* :

1. **Class Hierarchy:**

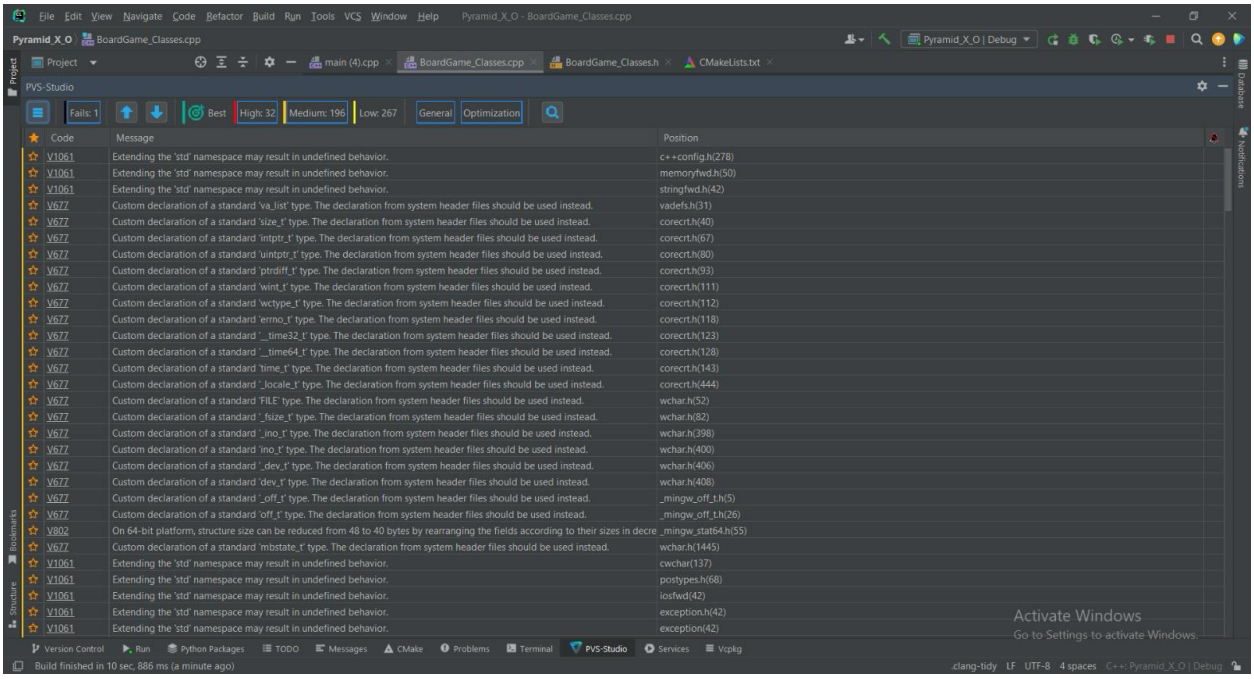
- The code defines a hierarchy of classes for a Four-in-a-Row X-O game.
- Classes include **Board**, **five_by_five_tictactoe_Board**, **Player**, **five_by_five_tictactoe_Player**, **RandomPlayer**, **GameManager**.
- The abstract base class **Board** provides an interface for game boards with pure virtual functions for updating the board, checking for a winner, a draw, and displaying the board.

2. **five_by_five_tictactoe_Board Class:**
    - This class represents the specific implementation of the game board for a traditional Tic-Tac-Toe (5x5) game.
    - This class contains **Is_Winner** function (different than is_winner) which is specific only for this game as it returns string and not Boolean. The function uses nested loops to first determine all three-in-a-row horizontally then vertically then diagonally from the left and diagonally from the right for each of X and O.
    - Once number of three-in-a-row Xs has been determined it is compared to the number of three-in-a-row Os.
    - Returns “X” if x is the winner, which then goes back to the game manager and declares X as a winner
    - Returns “O” if o is the winner, which then goes back to the game manager and declares O as the winner
    - If both numbers are the same, “Draw!” is returned and calls the is_draw function declaring a draw.
    - The **moves** function in this class determines if we have reached 24 moves, it’s specific only for this game and always returns false for all other games.
3. **Player Class:**
    - Represents a player with a name and a symbol (X or O).
    - Virtual function **get_move** is responsible for getting the player's move.
4. **five_by_five_tictactoe_Player Class:**
    - Derived from **Player**, this class is specific to the 5x5 tictactoe game.
    - Not much changes made, just the message that appears in terminal.
5. **RandomPlayer Class:**
    - Represents a computer player that makes random moves.
    - Overrides the **get_move** function to generate random moves.
6. **GameManager Class:**
    - Manages the flow of the game, including initializing the board and players, and running the game loop.
    - Has an extra if condition that allows the program to check the winner only after 24 moves.
7. **Main Function:**
    - In the **main** function, it creates instances of players based on user input.
    - It initializes the game manager with a 5x5 tictactoe board and the players and runs the game.

Bonus

PVS (Code Review and Code Quality)

Game 1



The new code after the edit on the warnings:

A) Player :

```
// Player.cpp

#include<iostream>
#include "XO_Classes.h"

Player::Player(char symbol) {
   this->symbol = symbol;
}

Player::Player(int order, char symbol) {
   std::cout << "Welcome player " << order << std::endl;
   std::cout << "Please enter your name: ";
   std::cin >> name;
   this->symbol = symbol;
}
```

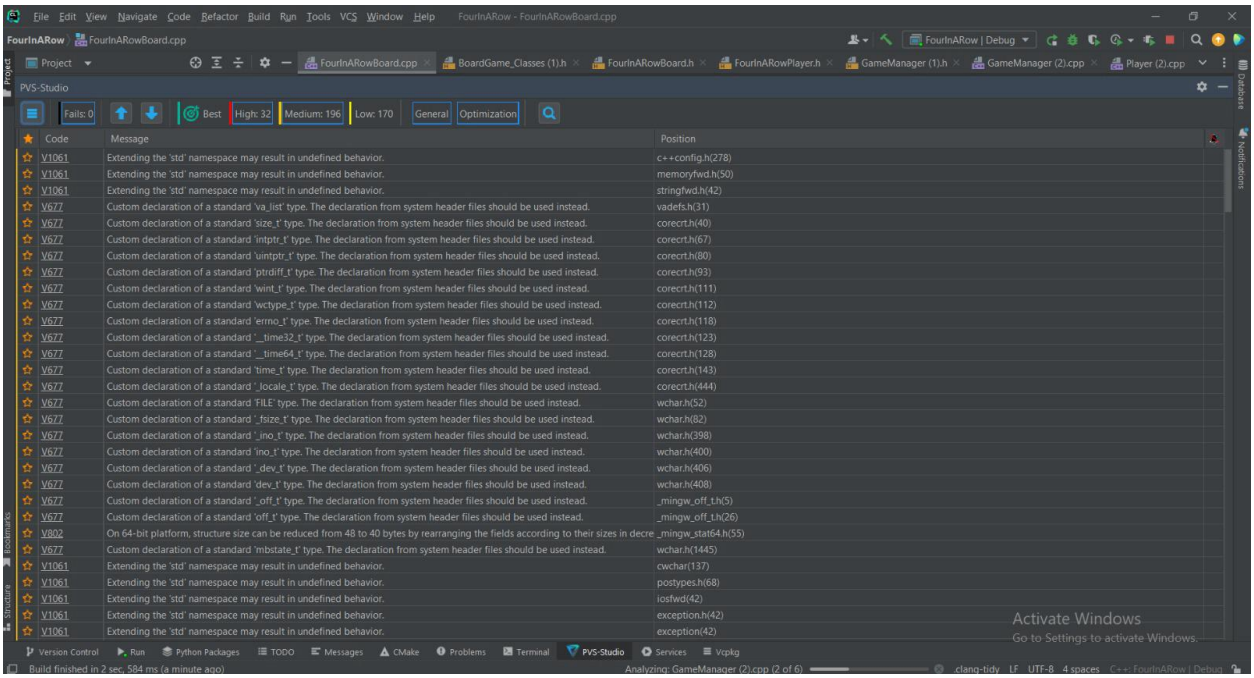
```
void Player::get_move(int &x, int &y) {
   std::cout << "\nPlease enter your move x and y (0 to 2) separated by spaces: ";
   std::cin >> x >> y;
}

std::string Player::to_string() {
   return "Player: " + name;
}

char Player::get_symbol() {
   return symbol;
}
```

B) Game manager :

```
// GameManager.cpp

#include <iostream>
#include "XO_Classes.h"

void GameManager::run() {
   Board x_o;
   int x, y, choice;
   Player *players[2];

   players[0] = new Player(1, 'X');

   std::cout << "Press 1 if you want to play with the computer: ";
   std::cin >> choice;

   if (choice != 1)
      players[1] = new Player(2, 'O');
   else
      players[1] = new ComputerPlayer('O', x_o);

   x_o.display_board();

   while (true) {
      for (int i : {0, 1}) {
         players[i]->get_move(x, y);

         while (!x_o.update_board(x, y, players[i]->get_symbol())) {
            players[i]->get_move(x, y);
```

```
        }

        x_o.display_board();

        if (x_o.is_winner()) {
            std::cout << players[i]->to_string() << " wins\n";
            delete players[0];
            delete players[1];
            return;
        }

        if (x_o.is_draw()) {
            std::cout << "Draw!\n";
            delete players[0];
            delete players[1];
            return;
        }
    }
  }
}
```

Game 2



The new code after the edit on the warnings:

A) Player :

```
// Player.cpp

#include<iostream>
#include "XO_Classes.h"

Player::Player(char symbol) {
   this->symbol = symbol;
}

Player::Player(int order, char symbol) {
   std::cout << "Welcome player " << order << std::endl;
   std::cout << "Please enter your name: ";
   std::cin >> name;
   this->symbol = symbol;
}

void Player::get_move(int &x, int &y) {
   std::cout << "\nPlease enter your move x and y (0 to 2) separated by spaces: ";
   std::cin >> x >> y;
}

std::string Player::to_string() {
   return "Player: " + name;
}

char Player::get_symbol() {
   return symbol;
}
```

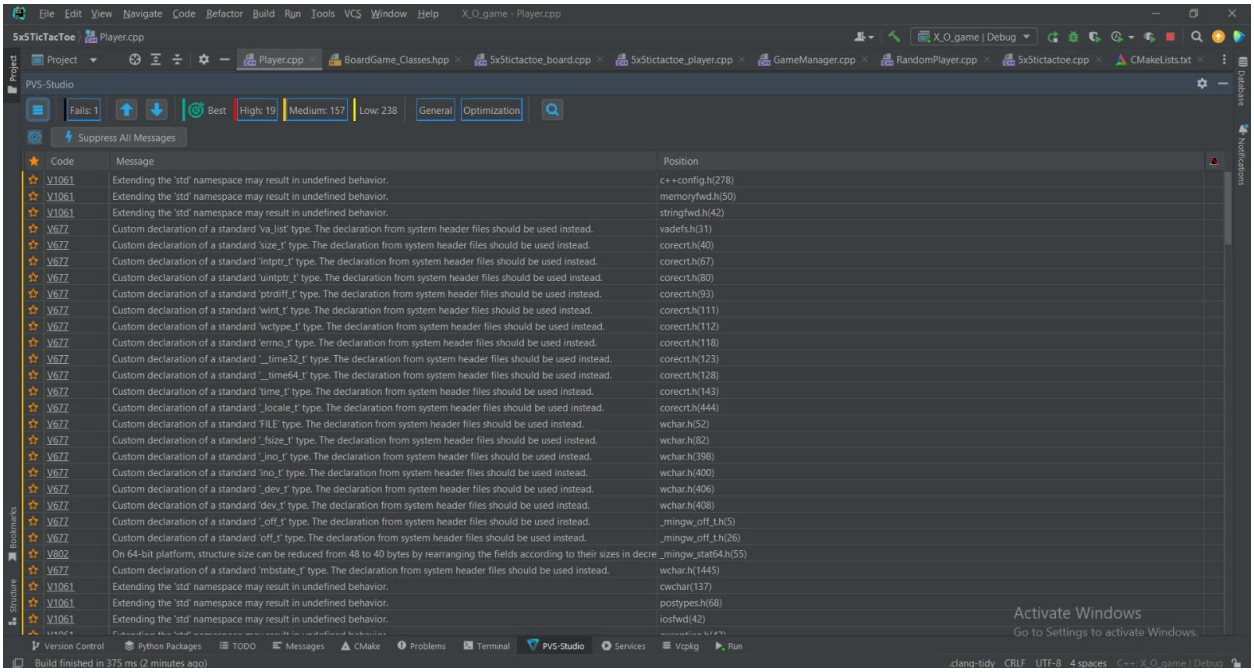
B) Game manager :

```
// GameManager.cpp

#include <iostream>
#include "XO_Classes.h"

void GameManager::run() {
   Board x_o;
   int x, y, choice;
```

```
    Player *players[2];

    players[0] = new Player(1, 'X');

    std::cout << "Press 1 if you want to play with the computer: ";
    std::cin >> choice;

    if (choice != 1)
        players[1] = new Player(2, 'O');
    else
        players[1] = new ComputerPlayer('O', x_o);

    x_o.display_board();

    while (true) {
        for (int i : {0, 1}) {
            players[i]->get_move(x, y);

            while (!x_o.update_board(x, y, players[i]->get_symbol())) {
                players[i]->get_move(x, y);
            }

            x_o.display_board();

            if (x_o.is_winner()) {
                std::cout << players[i]->to_string() << " wins\n";
                delete players[0];
                delete players[1];
                return;
            }

            if (x_o.is_draw()) {
                std::cout << "Draw!\n";
                delete players[0];
                delete players[1];
                return;
            }
        }
    }
}
```

-

Game 3



The new code after the edit on the warnings:

A) Player :

```
// Player.cpp

#include<iostream>
#include "XO_Classes.h"

Player::Player(char symbol) {
   this->symbol = symbol;
}

Player::Player(int order, char symbol) {
   std::cout << "Welcome player " << order << std::endl;
   std::cout << "Please enter your name: ";
   std::cin >> name;
   this->symbol = symbol;
}

void Player::get_move(int &x, int &y) {
   std::cout << "\nPlease enter your move x and y (0 to 2) separated by spaces: ";
   std::cin >> x >> y;
```

```
}

std::string Player::to_string() {
   return "Player: " + name;
}

char Player::get_symbol() {
   return symbol;
}
```

B) Game manager :

```
// GameManager.cpp

#include <iostream>
#include "XO_Classes.h"

void GameManager::run() {
   Board x_o;
   int x, y, choice;
   Player *players[2];

   players[0] = new Player(1, 'X');

   std::cout << "Press 1 if you want to play with the computer: ";
   std::cin >> choice;

   if (choice != 1)
      players[1] = new Player(2, 'O');
   else
      players[1] = new ComputerPlayer('O', x_o);

   x_o.display_board();

   while (true) {
      for (int i : {0, 1}) {
         players[i]->get_move(x, y);

         while (!x_o.update_board(x, y, players[i]->get_symbol())) {
            players[i]->get_move(x, y);
         }

         x_o.display_board();
```

```
        if (x_o.is_winner()) {
            std::cout << players[i]->to_string() << " wins\n";
            delete players[0];
            delete players[1];
            return;
        }

        if (x_o.is_draw()) {
            std::cout << "Draw!\n";
            delete players[0];
            delete players[1];
            return;
        }
      }
   }
}
```

C) Xo class.h

```
// XO_Classes.h

#ifndef _XO_CLASSES_H
#define _XO_CLASSES_H

#include <iostream>
#include <string>
#include <map>
#include <istream>
#include <ostream>
#include <cmath>
#include <regex>
#include <deque>
#include <set>
#include <algorithm>
#include <fstream>
#include <vector>
#include <sstream>
#include <iomanip>

class Board {
private:
   char board[3][3] = {{0}};
   int n_moves = 0;
```

```
public:
    bool update_board(int x, int y, char symbol);
    bool is_winner();
    bool is_draw();
    void display_board();
};

class Player {
private:
    std::string name;
    char symbol;

public:
    Player(char symbol);
    Player(int order, char symbol);
    virtual void get_move(int &x, int &y);
    std::string to_string();
    char get_symbol();
};

class ComputerPlayer : public Player {
private:
    Board &board;
    void best_move(int &x, int &y);
    int minimax(Board board1, bool isMax);

public:
    ComputerPlayer(char symbol, Board &board);
    void get_move(int &x, int &y) override;
};

class GameManager {
public:
    void run();
};

#endif
```

AI Player algorithm explanation

Code exploitation

1. **Board Class (Board.cpp and Board.h):**

- Represents a 3x3 game board for the X-O game.
- **update_board(int x, int y, char symbol)**: Updates the board with the player's move.
- **display_board()**: Displays the current state of the board.
- **is_winner()**: Checks if there is a winner on the board.
- **is_draw()**: Checks if the game is a draw.

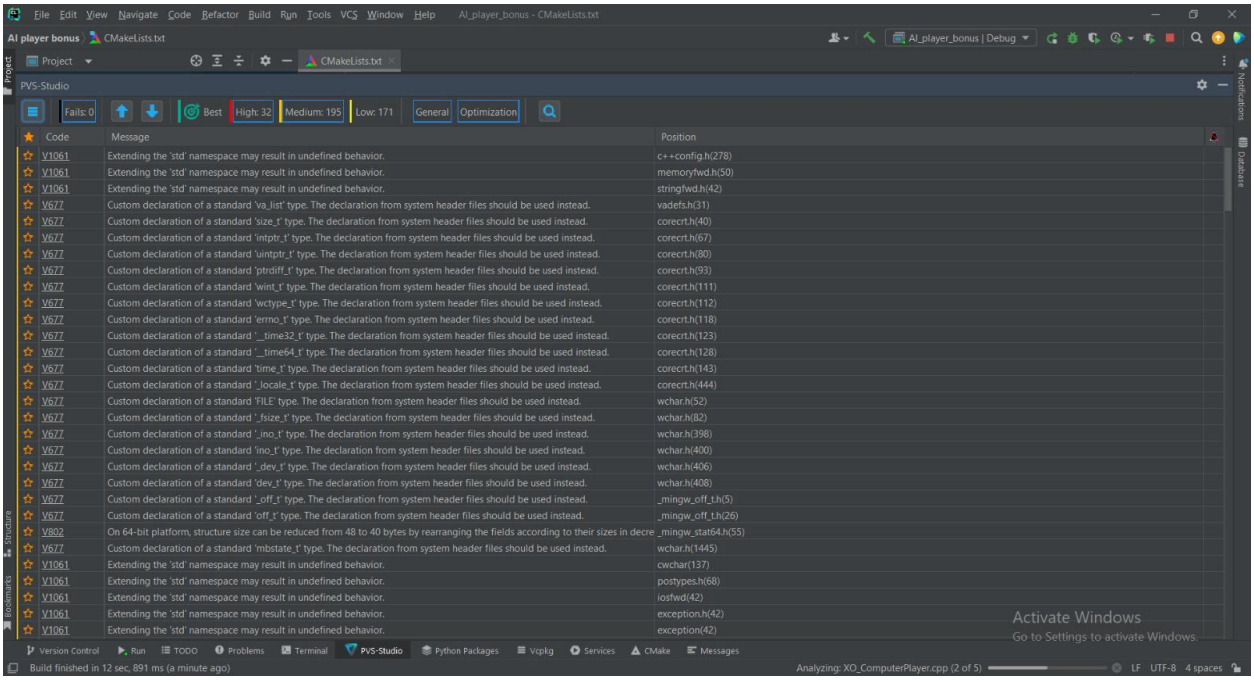
2. **Player Class (Player.cpp and Player.h):**
    - Represents a player with a name and symbol (X or O).
    - Two constructors for initiating players with and without an order.
    - **get_move(int &x, int &y)**: Virtual function to get the player's move.
    - **to_string()**: Returns player information as a string.
    - **get_symbol()**: Returns the player's symbol.
3. **ComputerPlayer Class (ComputerPlayer.cpp and ComputerPlayer.h):**
    - Inherits from Player, representing a computer player.
    - **best_move(int &x, int &y)**: Determines the best move using the minimax algorithm.
    - **minimax(Board board1, bool isMax)**: Implements the minimax algorithm for decision-making.
    - **get_move(int &x, int &y)**: Overrides the base class method to get a move for the computer.
4. **GameManager Class (XO_GameManager.cpp and XO_GameManager.h):**
    - Manages the flow of the X-O game.
    - **run()**: Creates the board and players, handles the game loop, and checks for a winner or draw.
5. **Main Program (main.cpp):**
    - Creates a GameManager object and runs the X-O game.
6. **XO_GameManager.cpp:**
    - Implements the **run()** function of the GameManager class.
    - Manages the game loop, player moves, and checks for a winner or draw.
7. **Player.cpp:**
    - Implements the Player class methods, including constructors, move input, and information retrieval.
8. **ComputerPlayer.cpp:**
    - Implements the ComputerPlayer class methods, including the minimax algorithm for decision-making.
9. **RandomPlayer.cpp and RandomPlayer.h (Not provided in the provided code):**
    - Mentioned in the initial description but not included in the code.
10. **Explanation of the Algorithm:**

- The computer player uses the minimax algorithm to determine the best move.
- The **best_move** function considers all possible moves and selects the one with the highest minimax score.
- The **minimax** function recursively evaluates possible outcomes of moves to determine the optimal move.

11. **GameManager::run() Function:**

- Manages the game flow, player moves, and checks for a winner or draw.
- Creates instances of the board and players, allowing the player to choose between playing against another player or a computer.

PVs studio



GUI

For GUI, we worked with C++ builder as it was the closest to C++ syntax and much easier.
Form1:

- First form that appears on screen, it asks the player to choose which game they would like to play

Form2:

- 3x3 TicTacToe
- Has its own form, c++ file, header file

Form3:

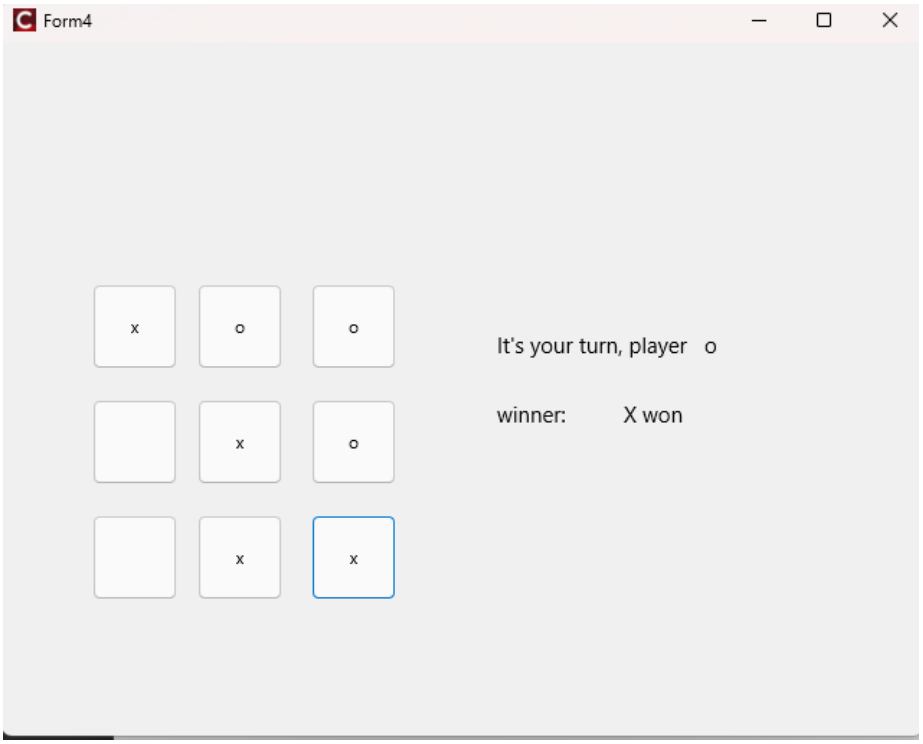
- Pyramid TicTacToe
- Has its own form, c++ file, header file

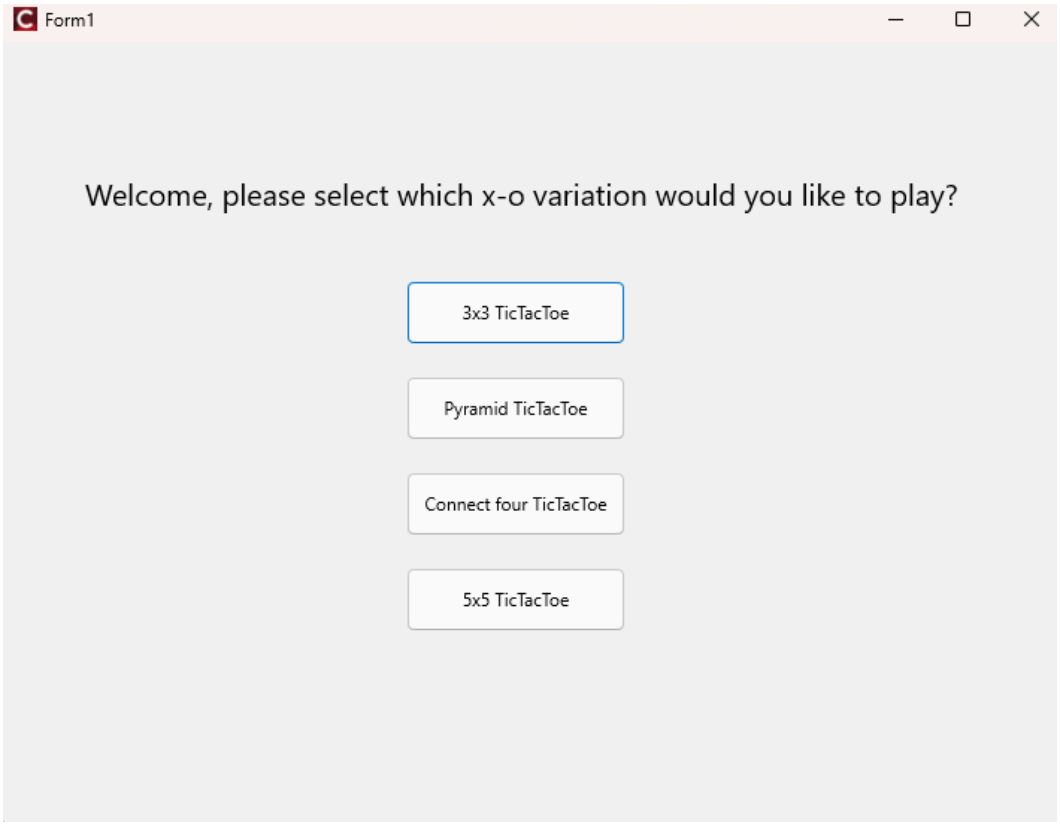
Form4:

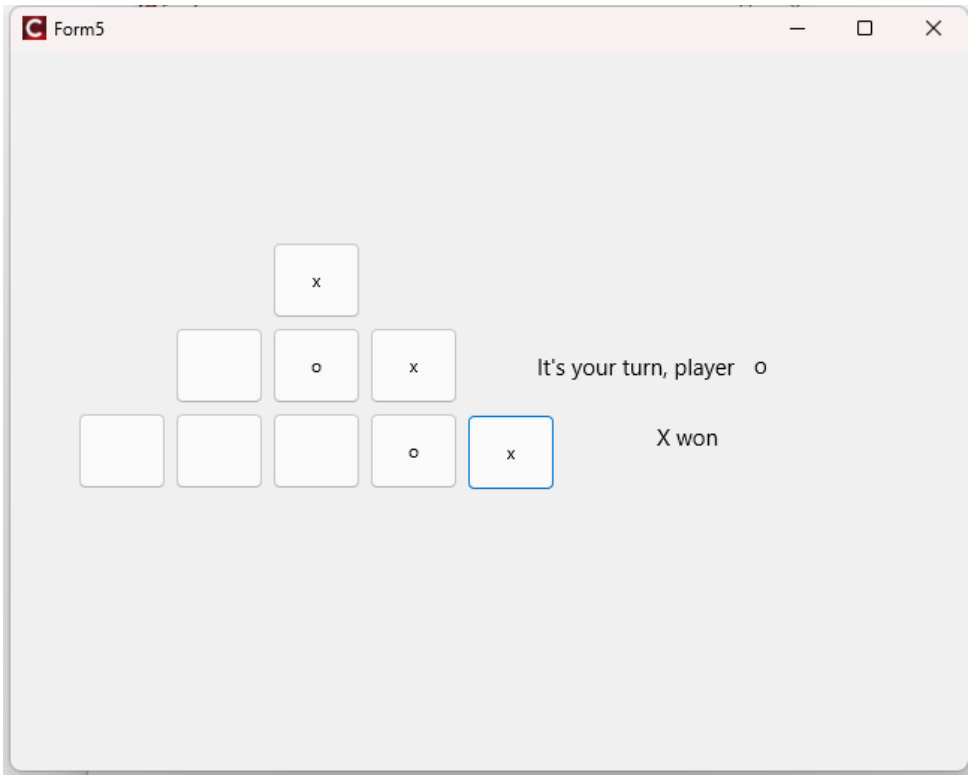
- Connect Four TicTacToe
- Has its own form, c++ file, header file

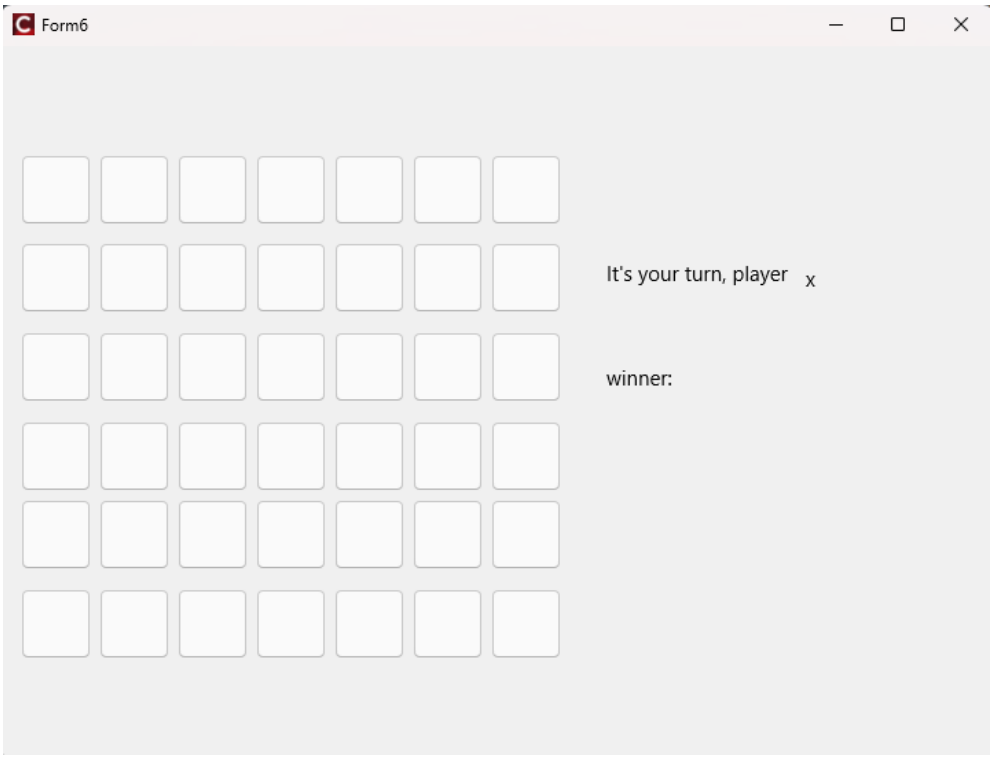
Form5:

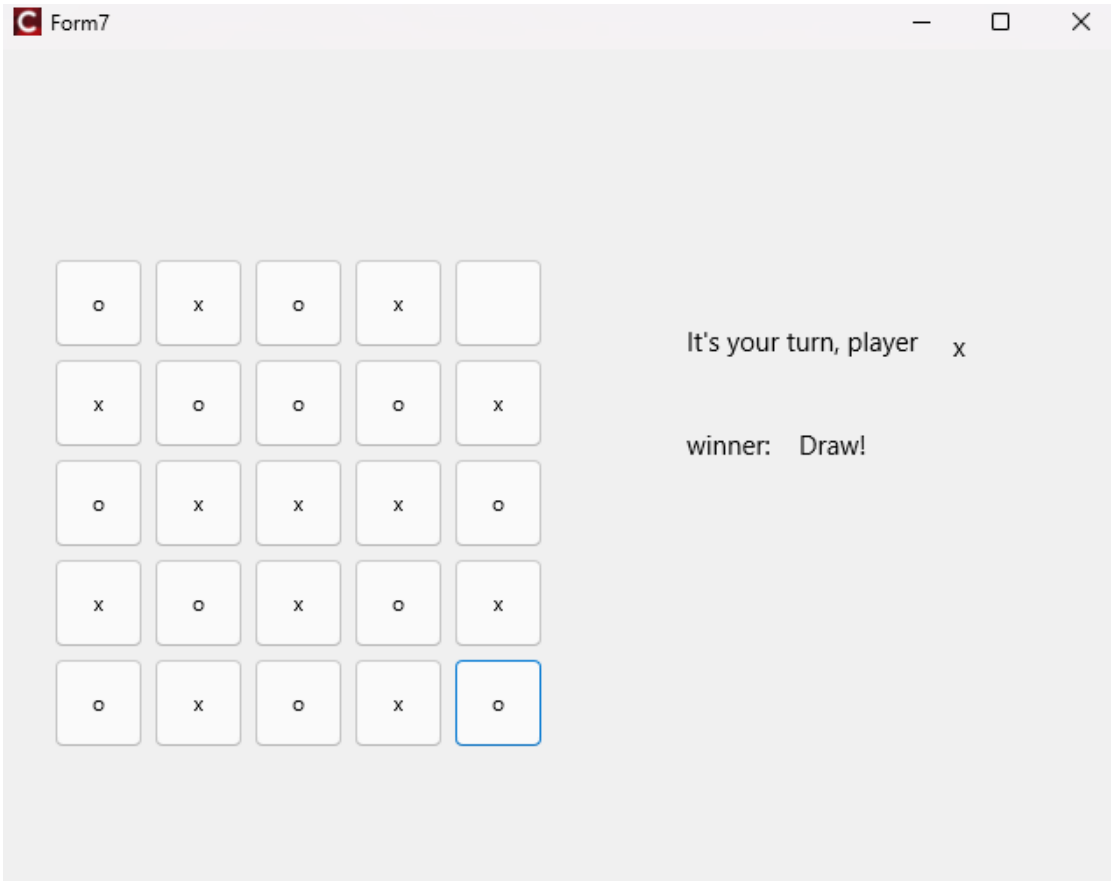
- 5x5 TicTacToe
- Has its own form, c++ file, header file


Form4
x
o
o
x
o
x
x
It's your turn, player o
winner: X won


Form1
Welcome, please select which x-o variation would you like to play?
3x3 TicTacToe
Pyramid TicTacToe
Connect four TicTacToe
5x5 TicTacToe


Form5
x
o
x
o
x
It's your turn, player o
X won


Form6
It's your turn, player x
winner:


Form7
o x o x
x o o o x
o x x x o
x o x o x
o x o x o
It's your turn, player x
winner: Draw!