Checkers game using OOPS in C++

PROJECT REPORT

1. Overview

This report details the development of a Checkers game implemented in C++. The project demonstrates key object-oriented programming concepts such as encapsulation, operator overloading, and dynamic memory management. It provides a two-player mode and a player-vs-computer mode for an interactive gaming experience.

2. Project Requirements and Objectives

The objectives of the Checkers game project were:

- To apply object-oriented programming principles such as encapsulation, abstraction, and operator overloading.
- To implement a two-player and player-vs-computer mode using a console-based interface. - To design an intuitive and manageable class structure for handling the board, players, and gameplay logic.

3. Design and Implementation

3.1 Class Structure

The Checkers game is designed using three main classes:

- Board: Manages the game board, pieces, and movement validation.
- Player: Represents a player, which can be either a human or a computer, and manages the player's moves.
- Game: Controls the game flow, switching between players and handling the main gameplay loop.

3.2 Object-Oriented Concepts

Encapsulation: Each class manages its data, exposing only necessary methods for interaction.

Operator Overloading:

- The Board class overloads the () operator for easy access to board pieces.
- The << operator is overloaded to facilitate printing the board directly with cout.
- The == operator is overloaded to compare two Board objects.

4. Code Implementation

4.1 Board Class

The Board class represents the 8x8 game board and handles piece placement, movement validation, and move execution. Here's a snippet showing part of the Board class:

```
class Board {
private:
    char board[BS][BS];
public:
    Board();
    void initialize();
    void print() const;
    bool bound(int x,int y) const;
    bool empty(int x,int y) const;
    char getp(int x,int y) const;
    void set(int x, int y, char get);
    char& operator()(int x, int y);
    friend ostream& operator<<(ostream& os, const Board& b);
    bool operator==(const Board& b) const;
    bool boardmove(int x1,int y1,int x2,int y2,char player);
    bool has_valid_moves(char player) const;
};
```

4.2 Player Class

The Player class represents a game player and handles the logic for move execution, distinguishing between human and computer moves.

```
class Player {
private:
    char get;
    bool iscomputer;
public:
    Player(char get,bool iscomputer);
    char getp() const;
    bool playermove(Board& board);
};
```

In the above class, playermove() function uses a simple algorithm where the computer always captures a piece if it is possible. If there is no possibility of capture the computer makes a random regular move (Note for teacher: Will try to improve this algorithm in the next version of the code).

4.3 Game Class

The Game class manages the main gameplay loop, switching turns between the two players and displaying the board after each move.

```
class Game {
```

```
private:
    Board board;
    Player* p1;
    Player* p2;
    Player* cp;
public:
    Game(int gm);
    ~Game();
    void sp();
    void play();
    bool check_game_over();
    void declare_winner();
};
```

4.4 Overloading Output(“<<”) operator

This function overloads the << operator to print a Board object, displaying the board's grid with column and row indices

```
ostream& operator<<(ostream& os, const Board& b) {
    os<<"  ";
    for(int i=0;i<BS;i++)
        os<<i<<" ";
    os<<endl;
    for(int i=0;i<BS;i++){
        os<<i<<" ";
        for(int j=0;j<BS;j++){
            os<<b.board[i][j]<<" ";
        }
        os<<endl;
    }
    return os;
}
```

4.5 Overloading “()” Operator

This function overloads the () operator for the Board class, allowing access to the board's cell at position (x, y) as if Board were a 2D array.

```
char& Board::operator()(int x, int y) {
    return board[x][y];
}
```

4.6 Overloading “==” Operator

This function overloads the == operator to compare two Board objects. It iterates through each cell in the board, checking if board[i][j] in the current object matches the corresponding cell in the given board b. If any cell differs, it returns false; if all cells match, it returns true.

```
bool Board::operator==(const Board& b) const {
    for(int i=0;i<BS;i++) {
        for(int j=0;j<BS;j++) {
            if(board[i][j]!=b.board[i][j]){
                return false;
            }
        }
    }
    return true;
}
```

4.7 Inheritance and Polymorphism

The Class PlayerBase is the base class. It is being inherited publicly by Player Class. The function getp() is declared virtual while playermove(Board & board) is declared pure virtual. The function playermove(Board & board) is being overridden in the publicly inherited Player Class.

```
class PlayerBase {
protected:
    char get;
    bool iscomputer;
public:
    PlayerBase(char get, bool iscomputer) : get(get),
iscomputer(iscomputer) {}
    virtual ~PlayerBase() {}
    virtual char getp() const { return get; }
    virtual bool playermove(Board& board) = 0;
};
class Player : public PlayerBase {
public:
    Player(char get, bool iscomputer);
    bool playermove(Board& board) override;
};
```

4.8 Exception Handling

The below application of Exception Handling ensures that the user inputs a valid game mode (between 1 and 3). If an invalid game mode is detected, it throws an exception, catches it, and reports the error. Otherwise, it initializes and starts the game.

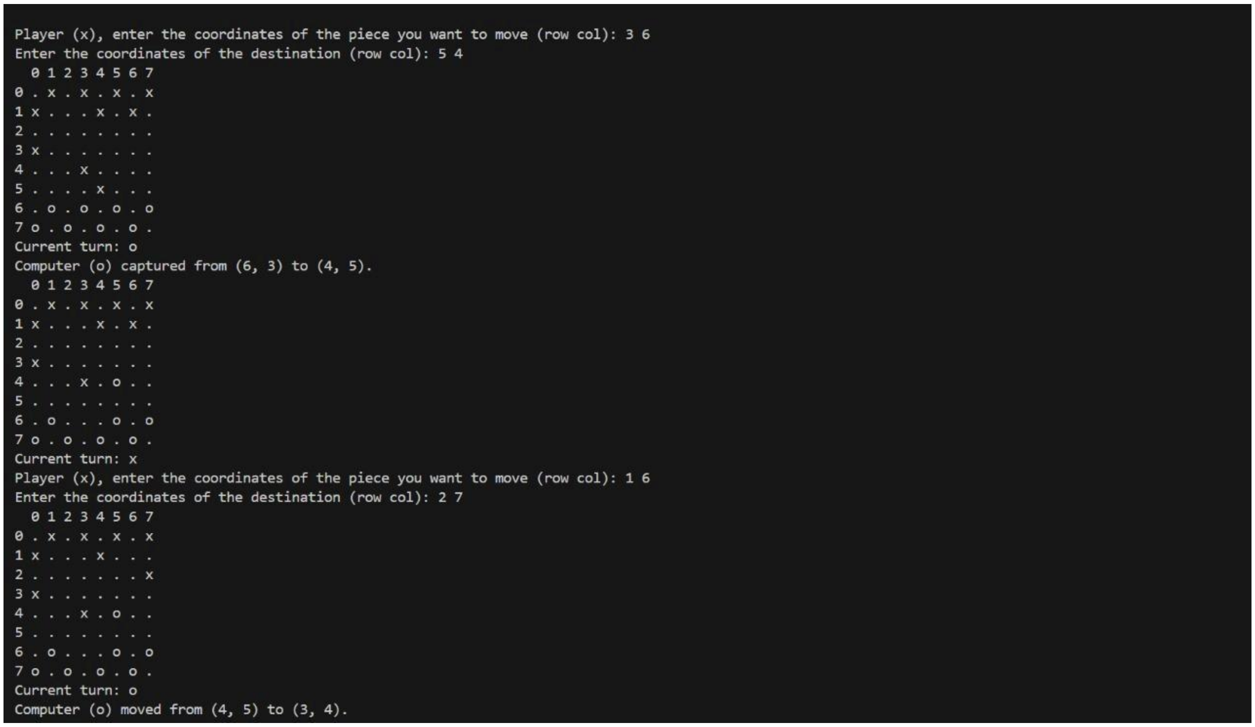
```
try {
        if (gm < 1 || gm > 3) {
            throw invalid_argument("Invalid game mode. Exiting.");
        }
        Game game(gm);
        game.play();
    } catch (const exception& e) {
        cout << "Error: " << e.what() << endl;
        return 1;
    }
```

6. Conclusion

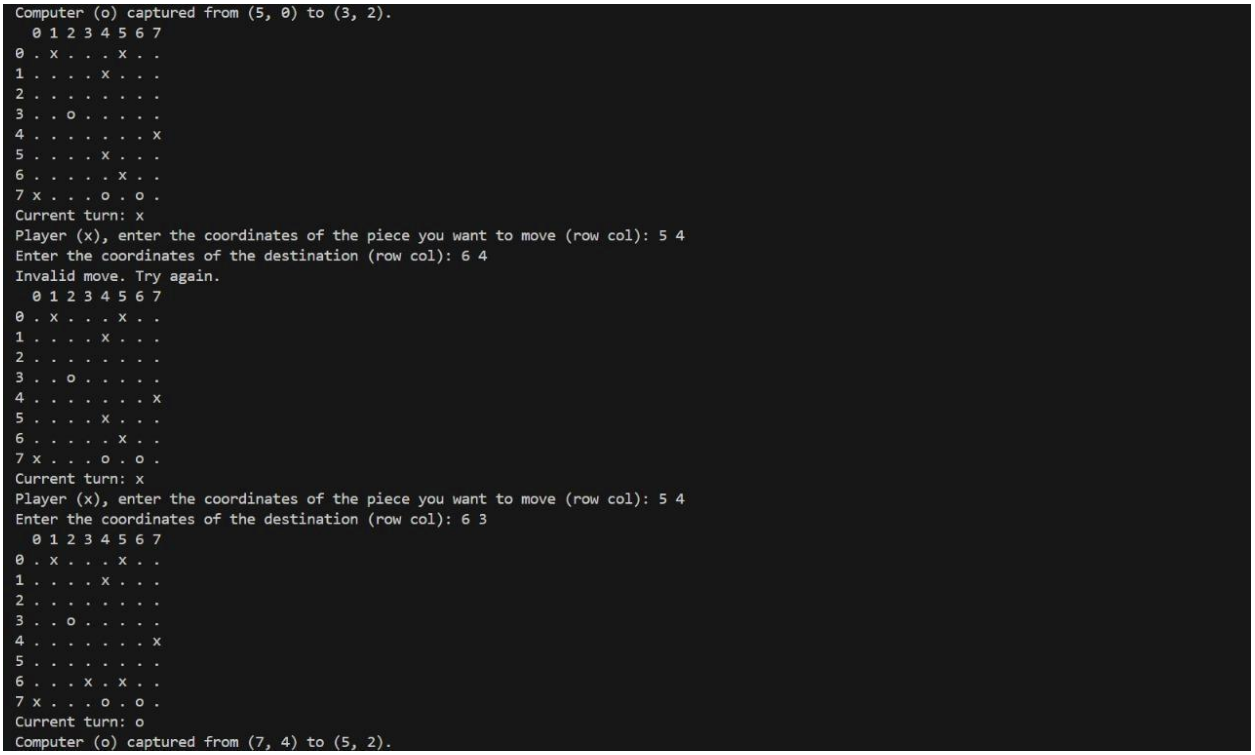
The Checkers game successfully demonstrates object-oriented principles in C++ by implementing encapsulation, operator overloading, and dynamic memory allocation. Future improvements could include graphical enhancements and additional difficulty levels for the computer opponent.

7. Outputs

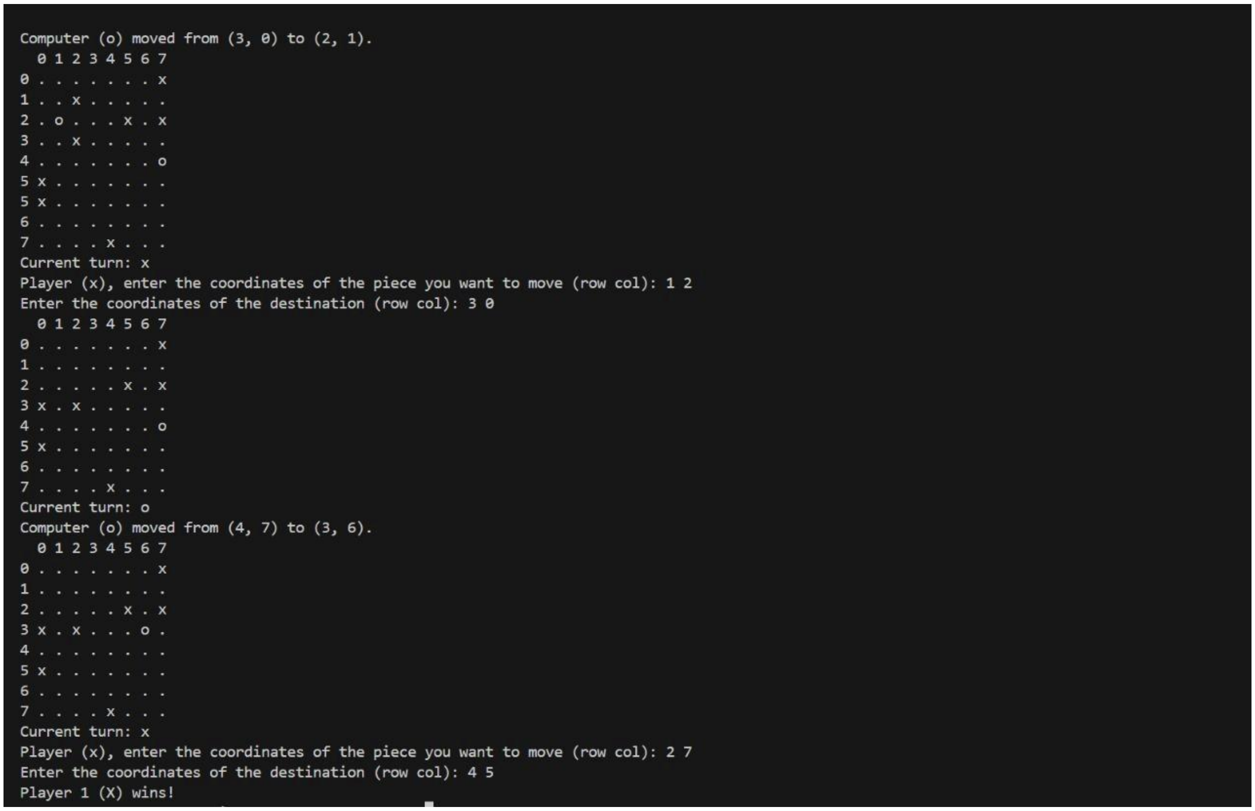
1. **The below screenshots are for PlayervsComputer:**



```
Player (x), enter the coordinates of the piece you want to move (row col): 3 6
Enter the coordinates of the destination (row col): 5 4
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . . . x . x .
2 . . . . . . . .
3 x . . . . . . .
4 . . . x . . . .
5 . . . . x . . .
6 . o . o . o . o
7 o . o . o . o .
Current turn: o
Computer (o) captured from (6, 3) to (4, 5).
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . . . x . x .
2 . . . . . . . .
3 x . . . . . . .
4 . . . x . o . .
5 . . . . . . . .
6 . o . . . o . o
7 o . o . o . o .
Current turn: x
Player (x), enter the coordinates of the piece you want to move (row col): 1 6
Enter the coordinates of the destination (row col): 2 7
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . . . x . . .
2 . . . . . . . x
3 x . . . . . . .
4 . . . x . o . .
5 . . . . . . . .
6 . o . . . o . o
7 o . o . o . o .
Current turn: o
Computer (o) moved from (4, 5) to (3, 4).
```

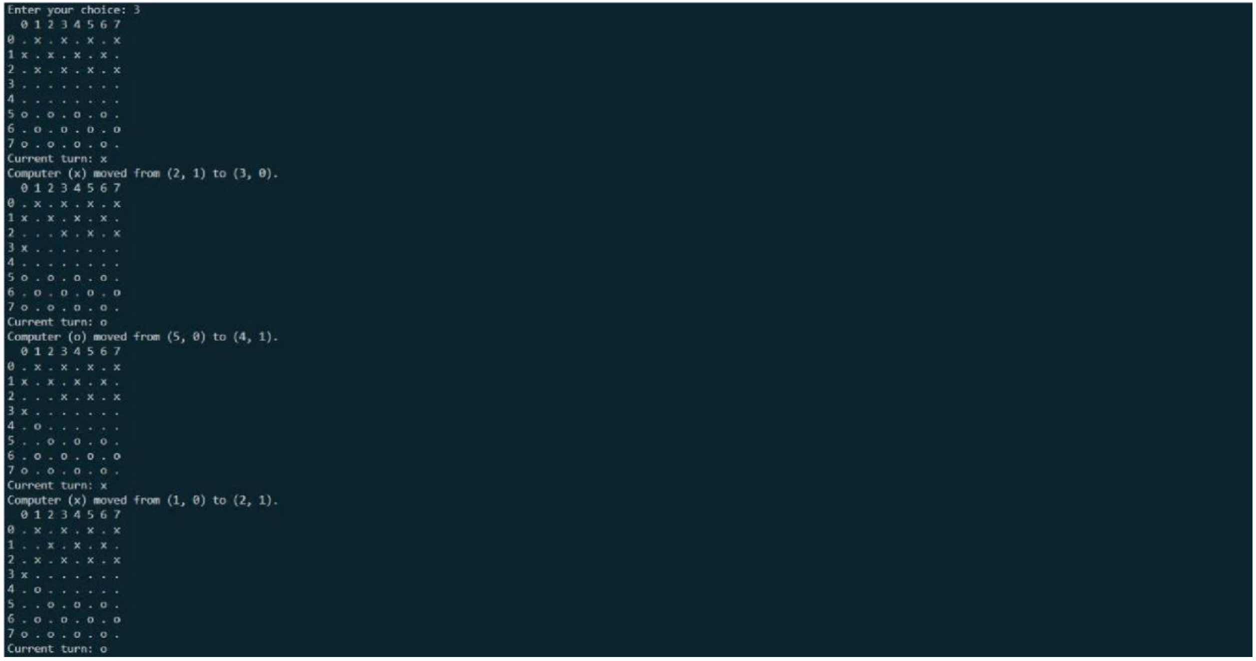


```
Computer (o) captured from (5, 0) to (3, 2).
  0 1 2 3 4 5 6 7
0 . x . . . x . .
1 . . . . x . . .
2 . . . . . . . .
3 . . o . . . . .
4 . . . . . . . x
5 . . . . x . . .
6 . . . . . x . .
7 x . . . o . o .
Current turn: x
Player (x), enter the coordinates of the piece you want to move (row col): 5 4
Enter the coordinates of the destination (row col): 6 4
Invalid move. Try again.
  0 1 2 3 4 5 6 7
0 . x . . . x . .
1 . . . . x . . .
2 . . . . . . . .
3 . . o . . . . .
4 . . . . . . . x
5 . . . . x . . .
6 . . . . . x . .
7 x . . . o . o .
Current turn: x
Player (x), enter the coordinates of the piece you want to move (row col): 5 4
Enter the coordinates of the destination (row col): 6 3
  0 1 2 3 4 5 6 7
0 . x . . . x . .
1 . . . . x . . .
2 . . . . . . . .
3 . . o . . . . .
4 . . . . . . . x
5 . . . . . . . .
6 . . . x . x . .
7 x . . . o . o .
Current turn: o
Computer (o) captured from (7, 4) to (5, 2).
```

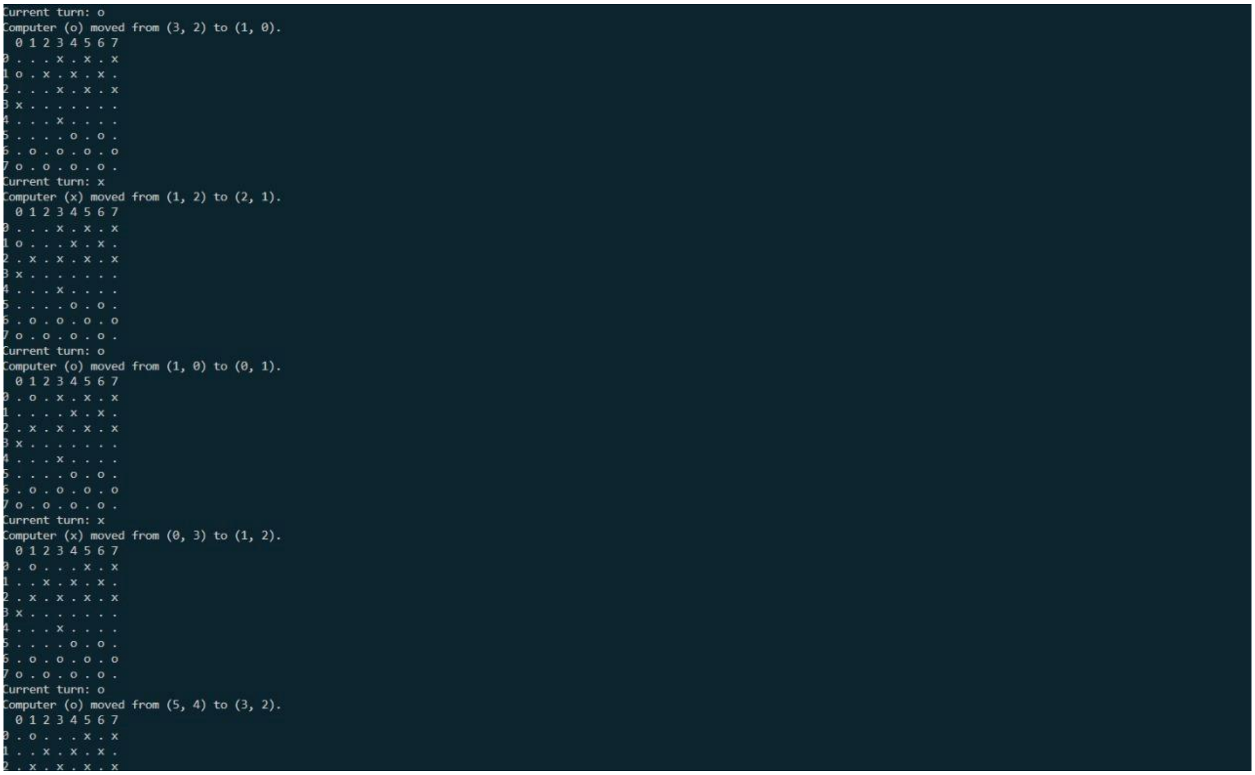


```
Computer (o) moved from (3, 0) to (2, 1).
  0 1 2 3 4 5 6 7
0 . . . . . . . x
1 . . x . . . . .
2 . o . . . x . x
3 . . x . . . . .
4 . . . . . . . o
5 x . . . . . . .
5 x . . . . . . .
6 . . . . . . . .
7 . . . . x . . .
Current turn: x
Player (x), enter the coordinates of the piece you want to move (row col): 1 2
Enter the coordinates of the destination (row col): 3 0
  0 1 2 3 4 5 6 7
0 . . . . . . . x
1 . . . . . . . .
2 . . . . . x . x
3 x . x . . . . .
4 . . . . . . . o
5 x . . . . . . .
6 . . . . . . . .
7 . . . . x . . .
Current turn: o
Computer (o) moved from (4, 7) to (3, 6).
  0 1 2 3 4 5 6 7
0 . . . . . . . x
1 . . . . . . . .
2 . . . . . x . x
3 x . x . . . o .
4 . . . . . . . .
5 x . . . . . . .
6 . . . . . . . .
7 . . . . x . . .
Current turn: x
Player (x), enter the coordinates of the piece you want to move (row col): 2 7
Enter the coordinates of the destination (row col): 4 5
Player 1 (X) wins!
```

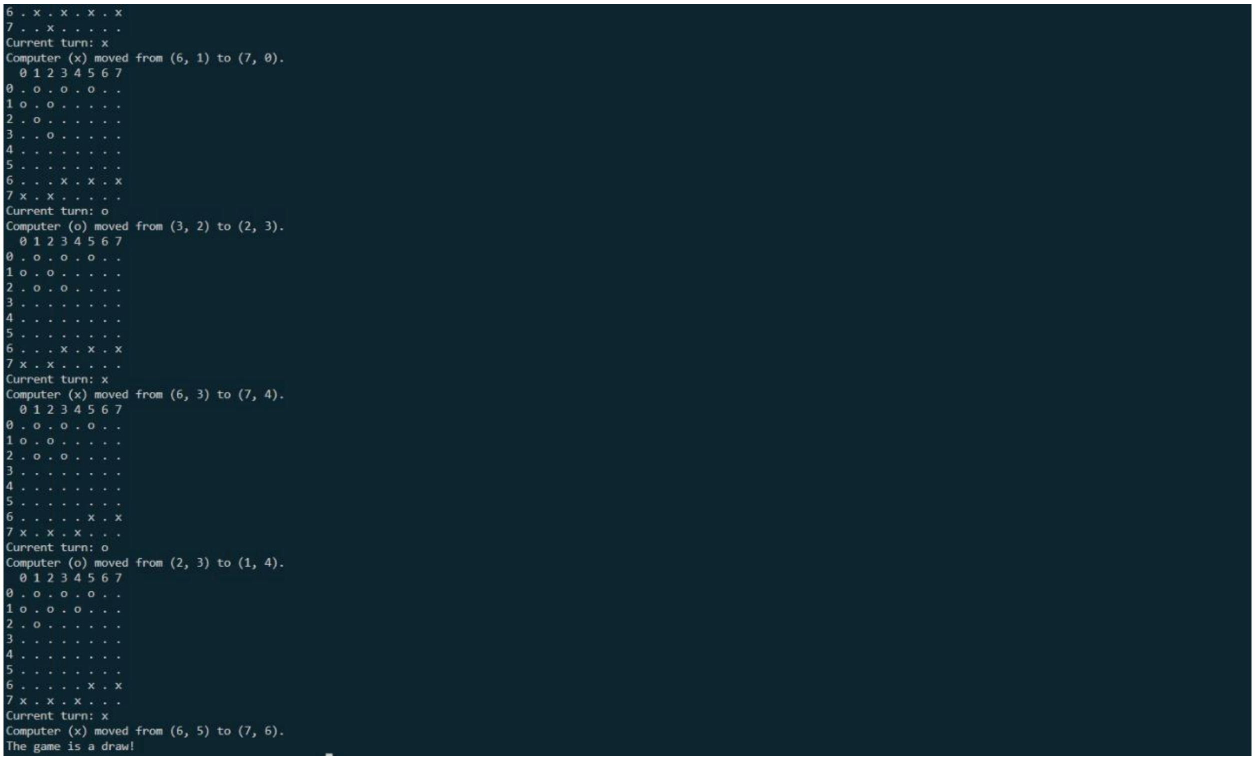
2. The below screenshots are for ComputervsComputer:



```
Enter your choice: 3
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . x . x . x .
2 . x . x . x . x
3 . . . . . . . .
4 . . . . . . . .
5 o . o . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: x
Computer (x) moved from (2, 1) to (3, 0).
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . x . x . x .
2 . . . x . x . x
3 x . . . . . . .
4 . . . . . . . .
5 o . o . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: o
Computer (o) moved from (5, 0) to (4, 1).
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 x . x . x . x .
2 . . . x . x . x
3 x . . . . . . .
4 . o . . . . . .
5 . . o . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: x
Computer (x) moved from (1, 0) to (2, 1).
  0 1 2 3 4 5 6 7
0 . x . x . x . x
1 . . x . x . x .
2 . x . x . x . x
3 x . . . . . . .
4 . o . . . . . .
5 . . o . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: o
```



```
Current turn: o
Computer (o) moved from (3, 2) to (1, 0).
  0 1 2 3 4 5 6 7
0 . . . x . x . x
1 o . x . x . x .
2 . . . x . x . x
3 x . . . . . . .
4 . . . x . . . .
5 . . . . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: x
Computer (x) moved from (1, 2) to (2, 1).
  0 1 2 3 4 5 6 7
0 . . . x . x . x
1 o . . . x . x .
2 . x . x . x . x
3 x . . . . . . .
4 . . . x . . . .
5 . . . . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: o
Computer (o) moved from (1, 0) to (0, 1).
  0 1 2 3 4 5 6 7
0 . o . x . x . x
1 . . . . x . x .
2 . x . x . x . x
3 x . . . . . . .
4 . . . x . . . .
5 . . . . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: x
Computer (x) moved from (0, 3) to (1, 2).
  0 1 2 3 4 5 6 7
0 . o . . . x . x
1 . . x . x . x .
2 . x . x . x . x
3 x . . . . . . .
4 . . . x . . . .
5 . . . . o . o .
6 . o . o . o . o
7 o . o . o . o .
Current turn: o
Computer (o) moved from (5, 4) to (3, 2).
  0 1 2 3 4 5 6 7
0 . o . . . x . x
1 . . x . x . x .
2 . x . x . x . x
```



```
6 . x . x . x . x
7 . . x . . . . .
Current turn: x
Computer (x) moved from (6, 1) to (7, 0).
  0 1 2 3 4 5 6 7
0 . o . o . o . .
1 o . o . . . . .
2 . o . . . . . .
3 . . o . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . x . x . x
7 x . x . . . . .
Current turn: o
Computer (o) moved from (3, 2) to (2, 3).
  0 1 2 3 4 5 6 7
0 . o . o . o . .
1 o . o . . . . .
2 . o . o . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . x . x . x
7 x . x . . . . .
Current turn: x
Computer (x) moved from (6, 3) to (7, 4).
  0 1 2 3 4 5 6 7
0 . o . o . o . .
1 o . o . . . . .
2 . o . o . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . x . x
7 x . x . x . . .
Current turn: o
Computer (o) moved from (2, 3) to (1, 4).
  0 1 2 3 4 5 6 7
0 . o . o . o . .
1 o . o . o . . .
2 . o . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . x . x
7 x . x . x . . .
Current turn: x
Computer (x) moved from (6, 5) to (7, 6).
The game is a draw!
```