Project 1

Tic Tac Toe Game

CIS-17C
Cynthia Hernandez
Due: 05/04/2024

Introduction

Title: 2 Player Tic Tac Toe

Game rules: Two players with two different markers take turn in filling empty squares in a 3x3 cell. The first player to mark three cells in a row with their mark, X or O, is the winner. Players may mark the cells in a horizontal or vertical pattern. If the all rows and columns of the cells are full, without three marks in a row presented, there is no winner.

EX:

X | O | X
—---------
X | X | O
—---------
X | O | O

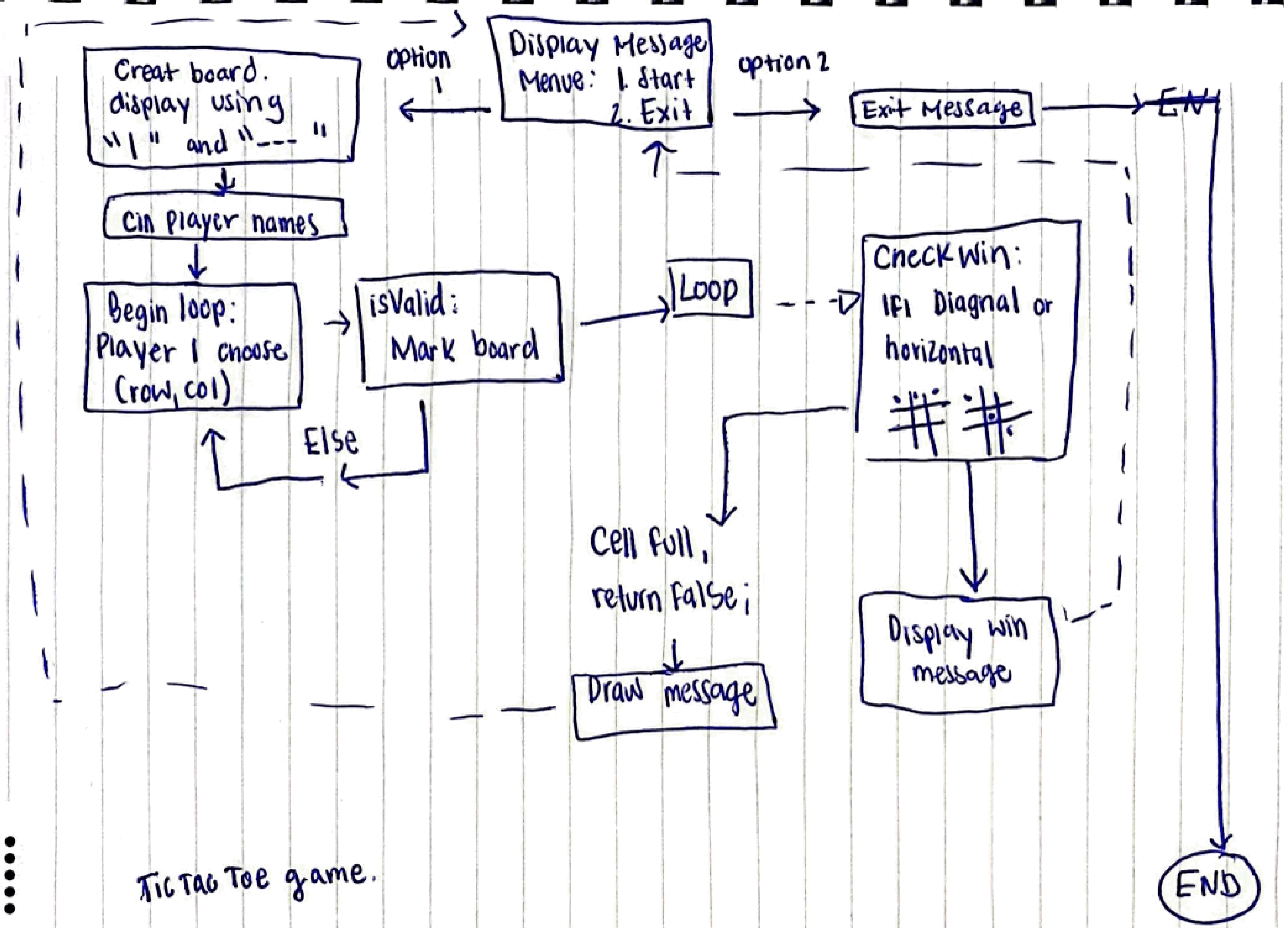
I chose this game because despite understanding it fully conceptually and in the real world, I wanted to see how the game would translate into code and how to integrate the graphics of each row and column of the typically drawn game.
I also chose this game because it is my absolute favorite way to pass time with a stranger or to teach to young children!

SUMMARY

Project size: 180 approx
Number of Variables: 10

Description

FLOW CHART



Tic Tac Toe game.

Check off list

Checkoff Sheet Contents

Sequences (At least 1)

List Sequence - I added a list called moveHistory, that allowed for tracking the moves of each player. This maintains all moves made throughout the game.

Associative Containers (At least 2)

Map container - my cells, or “board” used for the tic tac toe game stored value pairs (row, col).

Only one associative container is used.

3.Container adaptors (At least 2)

Queue - moveQueue manages the sequence of moves made. It stores the pairs of integers

Only one used. I was unable to fit another.

3.Algorithms (Choose at least 1 from each category)

1.Non-mutating algorithms

Non_mutating algorithms used were checkWin(), and the checkDraw() functions. They are not explicitly on the list but i believe they could be under the “search” for non- mutating.

Mutating algorithms

The code updates as the player enters their mark. This is not explicitly the fill function, or transform, but are similar. This happens in the start function.

Organization

None explicitly used. I realize that the game was far less complex than I had expected. I had a hard time successfully completing the checkoff sheet as I went down into the list.

PROGRAM

```
#include <iostream>
#include <map>
#include <list>
#include <queue>

using namespace std;

struct Player {
    char name[100];
    char mark;
    int wins;
    int moves;
};

class TicTacToe {
private:
    map<pair<int, int>, char> board;
    Player players[2];
    int currentPlayerIndex;
    bool gameOver;
    int numGames;
    int numDraws;
    list<pair<int, int> > moveHistory;
    queue<pair<int, int> > moveQueue;

public:
    TicTacToe() {
        players[0].mark = 'X';
        players[1].mark = 'O';
        currentPlayerIndex = 0;
        gameOver = false;
        numGames = 0;
        numDraws = 0;
    }
    void start() {
        displayMenu();
        char choice;
        cin >> choice;
        switch(choice) {
            case '1':
                startGame();
                break;
            case '2':
                displayStatistics();
                break;
            case '3':
                cout << "Exiting game! Thank You for playing. \n";
                exit(0);
            default:
                cout << "Invalid choice. Please try again.\n";
                start();
        }
    }

private:
    void displayMenu() {
        cout << "Welcome to Tic Tac Toe!\n";
        cout << "1. Start Game\n";
        cout << "2. View game history\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
    }

    void startGame() {
        initializeBoard();
        initializePlayers();
        moveHistory.clear();
        while (!moveQueue.empty()) {
            moveQueue.pop();
        }

        gameOver = false;
        numGames++;
        while (!gameOver) {
            printBoard();
            cout << players[currentPlayerIndex].name << ", enter your (row column): ";
            int row, col;
            cin >> row >> col;

            if (isValidMove(row, col)) {
                board[make_pair(row, col)] = players[currentPlayerIndex].mark;
                moveHistory.push_back(make_pair(row, col));
                moveQueue.push(make_pair(row, col));
                players[currentPlayerIndex].moves++;

                if (checkWin(players[currentPlayerIndex].mark)) {
                    gameOver = true;
                    players[currentPlayerIndex].wins++;
                    printBoard();
                    cout << players[currentPlayerIndex].name << " wins!" << endl;
                } else if (checkDraw()) {
                    gameOver = true;
                    numDraws++;
                    printBoard();
                    cout << "There is no winner. It's a draw!" << endl;
                } else {
                    currentPlayerIndex = (currentPlayerIndex + 1) % 2;
                }
            } else {
                cout << "Try again." << endl;
            }
        }
    }

    void initializeBoard() {
        board.clear();
        for (int i = 1; i <= 3; ++i) {
            for (int j = 1; j <= 3; ++j) {
                board[make_pair(i, j)] = ' ';
            }
        }
    }
```

```
    void initializePlayers() {
        cout << " ~~ This game requires two players ~~" << endl;
        cout << "Enter Player 1's name: ";
        cin >> players[0].name;
        cout << "Enter Player 2's name: ";
        cin >> players[1].name;

        players[0].wins = 0;
        players[1].wins = 0;
        players[0].moves = 0;
        players[1].moves = 0;
    }

    bool isValidMove(int row, int col) {
        return (row >= 1 && row <= 3 && col >= 1 && col <= 3 && board[make_pair(row, col)]
== ' ');
    }

    bool checkWin(char mark) {
        for (int i = 1; i <= 3; ++i) {
            if (board[make_pair(i, 1)] == mark && board[make_pair(i, 2)] == mark &&
board[make_pair(i, 3)] == mark) return true;
            if (board[make_pair(1, i)] == mark && board[make_pair(2, i)] == mark &&
board[make_pair(3, i)] == mark) return true;
        }
        if (board[make_pair(1, 1)] == mark && board[make_pair(2, 2)] == mark &&
board[make_pair(3, 3)] == mark) return true;
        if (board[make_pair(1, 3)] == mark && board[make_pair(2, 2)] == mark &&
board[make_pair(3, 1)] == mark) return true;
        return false;
    }

    bool checkDraw() {
        for (map<pair<int, int>, char>::iterator it = board.begin(); it != board.end(); ++it) {
            if (it->second == ' ') return false;
        }
        return true;
    }
    void printBoard() {
        cout << "-------------" << endl;
        for (int i = 1; i <= 3; ++i) {
            cout << "| ";
            for (int j = 1; j <= 3; ++j) {
                cout << board[make_pair(i, j)] << " | ";
            }
            cout << endl << "-------------" << endl;
        }
    }

    void displayStatistics() {
        cout << "Game Statistics:\n";
        cout << "Total Games Played: " << numGames << endl;
        cout << "Player 1 Wins: " << players[0].wins << endl;
        cout << "Player 2 Wins: " << players[1].wins << endl;
        cout << "Draws: " << numDraws << endl;
        cout << "Move History:\n";
        for (list<pair<int, int> >::iterator it = moveHistory.begin(); it != moveHistory.end(); ++it) {
            cout << "(" << it->first << ", " << it->second << ") ";
        }
        cout << endl;
    }
};

int main() {
    TicTacToe game;
    while (true) {
        game.start();
    }
    return 0;
}
```