Project 1

Title
Connect 4

Course
CIS-17C

Section
48209

Due Date
November 8, 2022

Author
Natalia Carbajal

Table of Contents

1 Introduction

Connect 4 has always been a fascinating and nostalgic game in my childhood. So I thought to myself, well, why not? Coding this game was a journey and I thought it would tie all the requirements together. I spent over 2 weeks before reaching the final version with 4 classes of what would be called the board, the game, the game state, and the players in the game.

2 Game Play and Rules

The primary goal of connect four is be the first player to get four “chips” of the same color/charcter in row either horizontal, vertical, or even diagonal, while preventing your opponent from doing the same. The Board consists of an 8 by 4 grid in which players can drop a marker, even if they must stack on top of another marker.

Objective:

To be the first player to connect 4 of the same markers in a row either vertically, horizontally, or diagonally.

How To Play:

• First, decide who goes first and which marker they will use (X or O)

• Players must alternate turns, and only one disc can be dropped in each turn.

• On your turn, drop one of your colored discs from the top into any of the available slots.

• The game ends when there is a 4-in-a-row or a stalemate

3 Development Summary

Line Size	818

While initially focusing on making sure the prompts from the program were clear and easy to understand, I also tried to lay out the game just like an arcade version. Two separate players should be able to easily navigate through the board and log their progress throughout, while having the program automatically detect wins in any direction. The program will be able to loop for another round or save players data for later use.

Layout

```
#include "Game.h"
#include "Board.h"

int main() {
    // Initalizes and runs game
    Game game;
    game.start();
    game.run();
}
```

This version includes the use of 4 header files and 4 source files, so that the main is extremely simplified and the data is passed through the various files. The player information (player.h/.cpp) has been stored in public and private classes and allows that information to be separate from the actual turn processes.

The use of header and source files resulted in the main becoming extremely brief. An instance of a class is created from the respective .h file and then passes that data into game.start() to start the actual game processes

```
class Board {
public:
  std::list<char> board; // Sequences: list
  std::map<std::string, char> player_markers;
  Board(const std::list<Player>& players);
public:
  bool mark(std::string name, int col);
  char vertWin();
  char hortWin();
  char diagWin();
  char win();
  void printBoard();
};

#endif // BOARD_H
```

board.h/cpp:

The board files keep track of the board size and display, wins, and each player's markers on the board by creating respective classes. The Cpp file Uses a constructer to create the board and uses a bidirectional iterator to check each column and determine where a marker can be placed. The board is then printed and wins are checked for in each possible direction

```
void MenuState::run() {
  while (true) {
    print_menu();
    string soption;
    int option = 3;
    getline(cin,soption);
    option = stoi(soption);
    if (option == 1) {
      game->gameState.push(new RuleState(game));
      game->gameState.top()->run();
    } else if(option == 2) {
      game->gameState.push(new CreateState(game));
      game->gameState.top()->run();
    } else {
      std::cout << "\nGoodbye!" << endl;
      break;
    }
  }
  game->gameState.pop();
}

RuleState::RuleState(Game* game) {
  this->game = game;
}
```

GameState.h/cpp:

The game state files use a series of glasses for each respective part of the game including the menu, creating a game, tracking the progress of the games, and the rules themselves. The cpp file executes each part of the game relevant to the established classes and utilizes text output save data to check if there are already players saved.

4 Output/Input

Upon starting the game the player is prompted to read the rules, play, or exit. If the player decides to check the rules they are displayed replicating the “board” that will be displayed. The player will then have the option to return to the menu or quit.

```
Welcome to Connect 4!
Enter the following options...
1. Rules
2. Play
3. Exit
```

If the program detects there is save data it will display a leader board where the players can select their names or create new players. The board is then displayed with the players name whose turn it is. By entering a number 0-7 your respective marker is dropped into the available slot on the board. After a win is detected the winning player will be displayed and a prompt will be displayed to start another game.

5 Checkoff Sheet

1. Container classes
    1. Sequences (At least 1)
        1. **list** (board.h 26 game.h 20)

        Used in public classes for player names and board characters

        2. **slist**
        3. **bit_vector** (board.h 27 game.h 20)

        Array used for board and players
    2. Associative Containers (At least 2)
        1. **set** (board.cpp 18) Used for markers (x's and 0's on board)
        2. **map** (board.h 27 board.cpp 20)

        Used for assigning markers to players

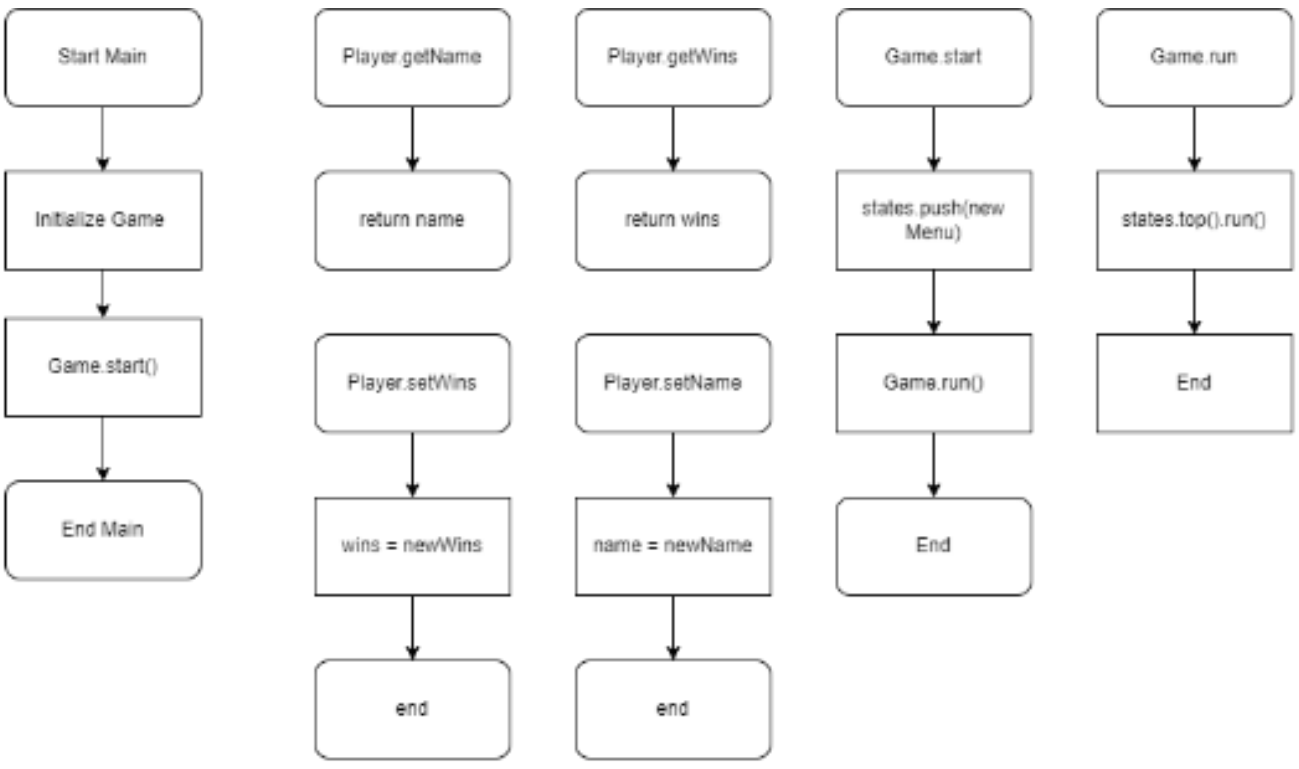
        3. **hash**
    3. Container adaptors (At least 2)
        1. **stack** (game.h 19) Used for tying in game state container adapter
        2. **queue** (gameState.cpp 235) Players go through queue based turn by turn
        3. **priority_queue**
2. Iterators
    1. Concepts (Describe the iterators utilized for each Container)
        1. Trivial Iterator
        2. Input Iterator
        3. Output Iterator
        4. Forward Iterator
        5. **Bidirectional Iterator** (board.cpp 26) Used to assign markers to players
        6. **Random Access Iterator** (gamestate.cpp 167) Uses duque in sort for players
3. Algorithms (Choose at least 1 from each category)
    1. Non-mutating algorithms
        1. **for_each** (board.cpp 35) Constructor used for board

2. find
3. count
4. equal
5. search

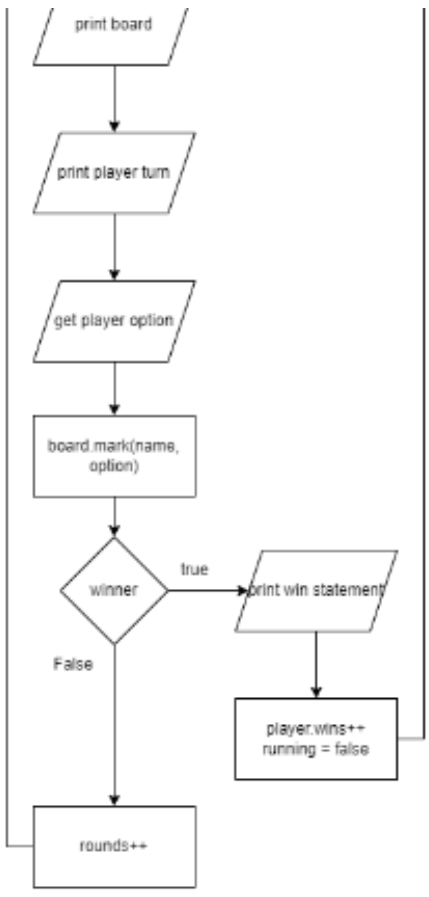
2. Mutating algorithms
    1. copy
    2. Swap
    3. Transform
    4. Replace
    5. **fill** (board.cpp 39)

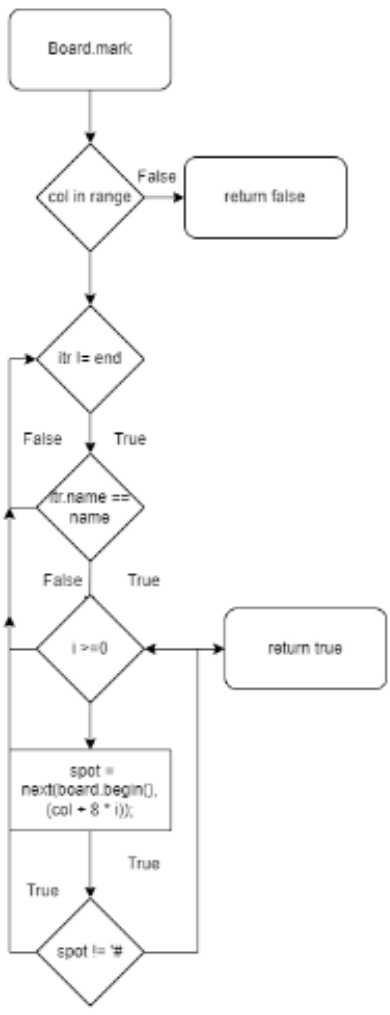
    Used to make board according to board size
    6. Remove
    7. Random_Shuffle
3. Organization
    1. **Sort** (GameState.cpp 167) Sorts players by win count
    2. Binary search
    3. merge
    4. inplace_merge
    5. Minimum and maximum

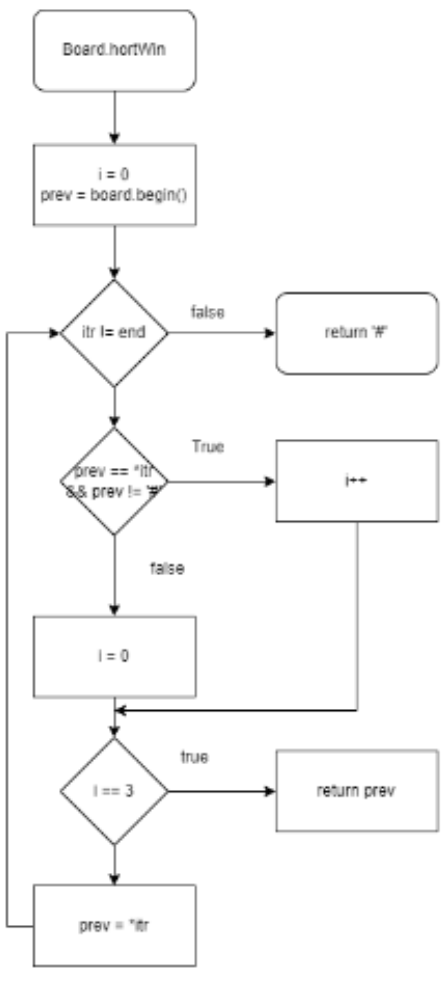
6 Flowchart

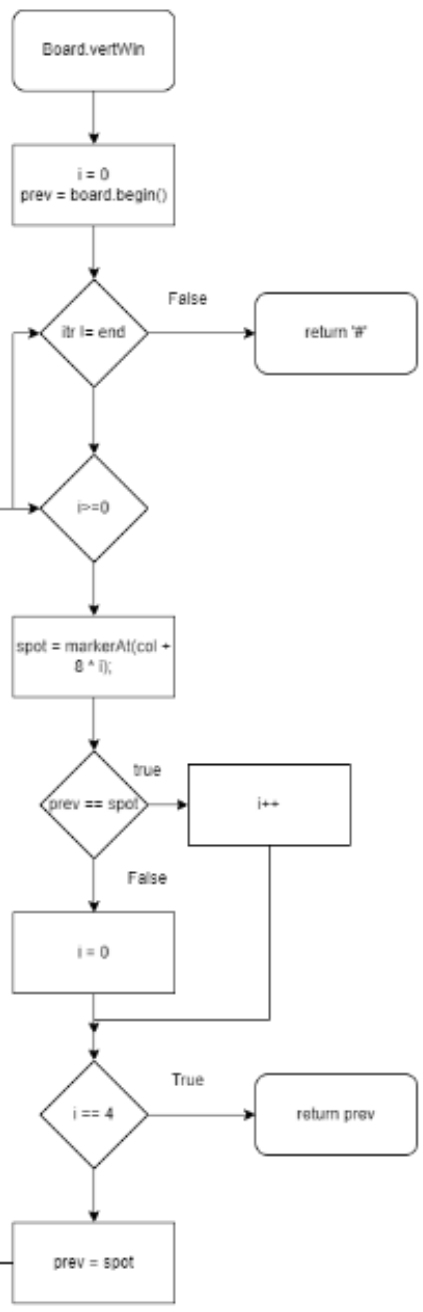

Start Main
Initialize Game
Game.start()
End Main
Player.getName
return name
Player.getWins
return wins
Game.start
states.push(new Menu)
Game.run()
End
Game.run
states.top().run()
End
Player.setWins
wins = newWins
end
Player.setName
name = newName
end
MenuState
game = nGame
End
RuleState
game = nGame
End
CreateState
game = nGame
End
PlayState
game = nGame
End

MenuState.run
break
true
game->gameState.pop()
End
print menu screen
get input option
3
option
1
gameState.push(new RuleState)
gameState.top().run()
2
gameState.push(new CreateState)
gameState.top().run()
CreateState.run
a
true
pop top stack
End
players.size
0
prompt Player name
player.pushback(Player(name))
sort players by win
prompt Player name
player.pushback(Player(name))
gameState.push(new RuleState)
gameState.top().run()
prompt 'Play again'
option
n
a
PlayState.run
Board board(players)
int rounds = 1
const int MAX_ROUNDS = 32
bool running = true
pop state
End
False
running
true
player in players
True
rounds>MAX_ROUNDS
False
Board::Board
for_each(begin,end, turn)
fill(board, '#')
End


print board
print player turn
get player option
board.mark(name, option)
winner
true
print win statement
False
player.wins++
running = false
rounds++


Board.mark
col in range
False
return false
itr != end
False
True
itr.name == name
False
True
i >=0
return true
spot = next(board.begin(), (col + 8 * i));
True
True
spot != '#


Board.hortWin
i = 0
prev = board.begin()
itr != end
false
return '#'
prev == *itr
&& prev != '#'
True
i++
false
i = 0
i == 3
true
return prev
prev = *itr


Board.vertWin
i = 0
prev = board.begin()
itr != end
False
return '#'
False
i>=0
spot = markerAt(col + 8 ^ i);
prev == spot
true
i++
False
i = 0
i == 4
True
return prev
prev = spot

Board.diagWin
int i = 0;
char prev = '#'
i <=5
False
i = 0
prev = '#'
True
False
j >=0
true
markerAt(i+8*i) == '#'
True
prev == markerAt(i+8*i)
True
i++
i = 0
i == 4
True
return prev
i <=5
False
return '#'
True
False
j >=0
markerAt(i+8*i) == '#'
True
prev == markerAt(i+8*j)
True
i++
i = 0
i == 4
True
return prev

Board.win
vertWin() != '#'
hortWin() != '#'
diagWin() != '#'