Project 1 - Connect 4

Michael Cooper

17C - 43484
Dr. Lehr
May 5, 2022

Introduction

I chose to code the game Connect 4 because I thought it would tie all the requirements together nicely and efficiently. I spent over 2 weeks before reaching the final build reaching 755 lines with 4 overarching classes for the board, game, game state, and players.

The repository can be found here: https://github.com/SouL909/Cooper_Michael_CSC17C_43484/tree/main/Projects/Project1

Approach To Development

Concepts

I used separate header files for each aspects of the game needed to make a complete build with all the necessary components. Each respective file manages a different part of the game including the visual representation of the board itself and each players markers as well as the state of the game and its players. The game state files tie in all the separate classes and organizes the phases of the game such as accessing the menu, reading the rules, creating a game, and keeping track of rounds/wins. In addition the wins and related data are saved via an output file and are able to be accessed when returning to the game after exiting and saving.

Version Control

The build took four versions to create with the first two taking a great deal of time to determine how to best separate each file and make relevant classes while encompassing all the needs of the project. I also ran into some difficulty getting the build to run with the restraints of my Mac and not having C++11 and had to do lots of troubleshooting to include all the requirements. Once finally getting the build to run and work correctly it was just a matter of optimization and making the inputs and outputs user friendly.

Game Rules

Game Title: Connect Four

The classic table top game which involves trying to successfully complete a consecutive sequence of four markers, 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.

Description of Code

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 playersdata for later use.

Layout:

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 completely separate from the actual turn processes.

A Look at the Main:

```cpp
#include "Game.h"
#include "Board.h"
//Starts the game
int main() {
  Game game;
  game.start();
}
```

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.

Header and Source Files Examples:

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.

```cpp
const int BOARD_SIZE = 8*4;
//Class declaration, keeps track of each individual tile for X's or O's
//Also keeps track of which player is X or O
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
```

```cpp
//Constructor to create board
Board::Board(const std::list<Player>& players) {
  _player_markers = &player_markers;
  for_each(players.cbegin(), players.cend(), fe);
  _player_markers = nullptr;
  board.resize(BOARD_SIZE);
  fill(board.begin(), board.end(), *MARKERS.cbegin()); // Mutating Algorithm: f
}

//Checks each column of array and sees where it can place marker
bool Board::mark(string name, int col) {
  if(col < 0 && col >= 8) {
    return false;
  }
  for(std::map<string, char>::iterator itr = player_markers.begin(); itr != pla
    if(itr->first == name) {
      for(int i = 3; i >= 0; i--) {
        auto spot = next(board.begin(), (col + 8 * i));
        if(*spot != '#')
          continue;
        else {
          *spot = itr->second;
          break;
        }
      }
      return true;
    }
  }
  return false;
}
```

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.

```
class GameState {
public:
  virtual void run() = 0;
};

class MenuState : public GameState {
private:
  Game* game;
public:
  MenuState(Game* game);
  void run();
};

class CreateState : public GameState {
private:
  Game* game;
public:
  CreateState(Game* game);
  void run();
};

class PlayState : public GameState {
private:
  Game* game;
public:
  PlayState(Game* game);
  void run();
};

class RuleState : public GameState {
private:
  Game* game;
public:
  RuleState(Game* game);
  void run();
};
#endif // GAMESTATE_H
```

```
// Makes a copy of the save file
// In order to update each player's
// win counter
fstream myfile ("players.txt", std::fstream::in | std::fstream::out);
if(myfile.is_open()) {
  deque<Player> temp;
  string line;
  int i = 0;

  deque<string> names;
  deque<int> wins;

  // Goes through the file
  // and searches for names and wins
  while(getline(myfile, line)) {
    if(line.empty()) continue;
    if(i % 2 == 0) {
      names.push_back(line);
    } else {
      wins.push_back(stoi(line));
    }
    i++;
```

Sample Input/Output

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

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.

```
Leaderboards and selection
----------------------------------------------------
(1) Player: Michael Wins: 1
----------------------------------------------------
----------------------------------------------------
(2) Player: Michael Wins: 1
----------------------------------------------------
----------------------------------------------------
(3) Player: Jacob Wins: 0
----------------------------------------------------
----------------------------------------------------
(4) Player: Jacob Wins: 0
----------------------------------------------------
Would you like to create new players? (y/n)
```

```
Would you like to create new players? (y/n)
n
Player 1 select your profile: 1
Player 2 select your profile: 3
Playing...

# # # # # # # #
# # # # # # # #
# # # # # # # #
# # # # # # # # Michael's turn!
Enter a number from 0-7
```

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. The different win scenarios are displayed below.

```
x # # # # # # #
x # # # # # # o
x # # # # # # o
x # # # # # # o Michael WINS!
Play again? (y/n):
```

```
# # # x # # # #
# # x o x # # #
# x o o x # # #
x o o x o # # # Michael WINS!
Play again? (y/n):
```

```
# # # # # # # #
# # # # # # # #
o o o # # # # #
x x x x # # # # Michael WINS!
Play again? (y/n):
```

Checkoff Sheet

1. Container classes
    1. Sequences (At least 1)
        1. <u>list</u> (board.h 26 game.h 20)
        Used in public classes for player names and board characters
        2. <u>slist</u>
        3. <u>bit_vector</u> (board.h 27 game.h 20)
        Array used for board and players
    2. Associative Containers (At least 2)
        1. <u>set</u> (board.cpp 18)
        Used for markers (x's and 0's on board)
        2. <u>map</u> (board.h 27 board.cpp 20)
        Used for assigning markers to players
        3. <u>hash</u>
    3. Container adaptors (At least 2)
        1. <u>stack</u> (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

Flowchart (other docs attached separately)


Start Main
Initialize Game
Game.start()
End Main
Player.getName
return name
Player.getWins
return wins
Player.setWins
wins = newWins
end
Player.setName
name = newName
end
Game.start
states.push(new Menu)
Game.run()
End
Game.run
states.top().run()
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
option
3
1
2
gameState.push(new RuleState)
gameState.top().run()
gameState.push(new CreateState)
gameState.top().run()
CreateState.run
true
players.size
0
prompt Player name
player.pushback(Player(name))
prompt Player name
player.pushback(Player(name))
sort players by win
gameState.push(new RuleState)
gameState.top().run()
prompt 'Play again'
option
n
a
a
pop top stack
End
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
round>MAX_ROUNDS
False
print board
print player turn
get player option
board.mark(name, option)
winner
true
print win statement
False
player.wins++
running = false
rounds++
Board::Board
for_each(begin,end, turn)
fill(board, '#')
End
Board.mark
col in range
False
return false
itr != end
True
False
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 l = 0;
char prev = '#'
i <=5
False
l = 0
True
False
j >=0
true
markerAt(l+8*i) == '#'
True
prev == markerAt(i+8*j)
True
i++
l = 0
l == 4
True
return prev
prev = '#'
i <=5
False
return '#'
True
False
j >=0
markerAt(l+8*i) == '#'
True
prev == markerAt(i+8*j)
True
i++
l = 0
l == 4
True
return prev
Board.win
vertWin() != '0'
hortWin() != '0'
diagWin() != '#'
return vertWin()
return hortWin()
return diagWin()