Connect 4 C++ Game

Introduction

For this project I chose the game Connect 4 with the classix 7x6 board. I spent about 3 hours coding the game with no regard to the requirements. An additional 2 hours was spent fixing logical errors when cleaning up the code and 2 hours trying to implement the requirements into my project. I also spent 10 minutes transferring my code to NetBeans. In total, I spent 7 hrs and 10 minutes on the project.

Approach to Development

The first step I took was displaying a basic board and the rules for the game. I set up the initial menu and request for input from the user. After I set up the board, I converted it into a map of pair<int,int> and char type to meet the project requirements.

Afterwards, I began working on non-mutating functions inside of main(). I made note of which functions could be used in a class and which could be used by the main program.

Lastly, I worked on implementing the algorithm for winning the game or in case of a draw. I added a simple random generator to choose which player would go first and added an option to play against the computer. I then moved all functions into a base class and added the rest of the required implementations to my project.

Version Control

The version control for this project was kept in separate files in a different IDE from NetBeans. VS Code Studio was used until the class version of the game was finished. The last version of the game was uploaded to github at: https://github.com/vdd0/project1.git

Game Rules

The game rules of Connect 4 is to get a linear combination of 4 unbroken checker pieces on the board. Each player takes turns to drop their checker into a column of their choice. If a player gets 4 checkers of the same color to line up horizontally, vertically or diagonally then the game ends. Otherwise, if all slots are filled then the game ends at a draw

Description of Code

Only one class was used for the game. Most of the game's functionality was called through the playTurn member function in the class. Helper functions outside of main were called to print the game history and the leaderboard.

The total lines of code in the project is 503 lines. With 401 lines of code and 102 of comments.

Class:

- Connect4:
    - Manages the connect4 game by using PlayTurn() function to prompt for user input and end() function to stop the game in main();

User Input Validation:

Throughout the game, wherever a user was required to choose an option, I added a validation criteria that required the user to retry until their input was successful.

Check Off Sheet:

Containers, iterators, and algorithms implementation Check off Sheet

CONTAINER CLASS

SEQUENCES (at least 1)

- ☑ ~~List~~
- ☐ Slist
- ☐ Bit_vector

ASSOCIATIVE CONTAINERS (at least 2)

- ☐ Set
- ☑ ~~Map~~
- ☐ Hash

CONTAINER ADAPTORS (at least 2)

- ☑ ~~Stack~~
- ☑ ~~Queue~~
- ☐ Priority_queue

ITERATORS

CONCEPTS (describe the iterators utilized for each container)

- ☑ ~~Trivial iterator~~
- ☑ ~~Input iterator~~

- ☑ ~~Output iterator~~

ALGORITHMS (choose at least 1 from each category)

NON-MUTATING ALGORITHMS

- ☑ ~~Find~~

MUTATING ALGORITHM

- ☑ ~~Swap~~
- ☐ Fill

Container Classes

- ➢ Sequences
  - ■ list
    - Used in print helper function to initialize a list with all scores. The list was used as an argument for printLeaderboard and sorted.
- ➢ Associative Containers
  - ■ Map
    - Used in connect4 class to store the board and to store the rows available with each column.
- ➢ Container Adaptors
  - ■ Stack
    - Used in the helper function printLeaderboard() to reverse the sort from list (ascending) to descending.
  - ■ Queue

- Used in main to store the board from the game and the associated winner.

Iterators

- Trivial Iterator: Used in setBoard to set the board and also to print the board.
- Input Iterator: Used in updateBoard to find an available slot.
- Random Access Iterator: Used in printLeaderboard for accessing the winner and count.

Algorithms

- ➢ Non-mutating algorithms
  - find
    - Used in helper function print(queue<pair<map<pair<int,int>,char>,string>>) to find the associated score with the person and update it.
- ➢ Mutating algorithms
  - swap
    - Used in playTurn to swap the players and to change the checkerPieces from its initialization once user input is different.
- ➢ Organization
  - Sort
    - Used to sort the list in the helper function of printLeaderboard()

Documentation of Code

Class UML

```
connect4
-static const int max_rows
-static const int max_cols
-pair<string,char> player;
-pair<string,char> opponent;
-map<int,int> slots;
-int count;

Public:
-string winner;
-string opp;
-map<pair<int,int>, char> board;
-bool setUsers;
-connect4()
-connect4(string,string)
-void setBoard();
-void playTurn();
-void updateBoard();
-bool end();
-bool checkOutcome();
-void displayWin();
-void print();
-void displayMenu();
-int computerTurn();
-int randomTurn()
```

```
Helper Functions:

-void printMap(map<pair<int,int>,char>b)
-void printLeaderboard(map<string,int>b,
list<int>score)
-void unsortedLead(map<string,int>b)
-void
print(queue<pair<map<pair<int,int>,char>,stri
ng>> h)
```

PsuedoCode:

```
//psuedo code
// display an empty table of 7x6
//  have one player choose whether they are X or O
// set the color/option
// ask if they want to choose who goes first, or randomize
//  each player takes turn
// validate input
// display board after each turn
// player with 4 in a diag/row/col wins
// end game
// ask if they'd like to play again or quit
```