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 1 hour trying to implement the requirements into my project. I also spent 10 minutes transferring my code to NetBeans. In total, I spent 6 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.

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 called game and only called two functions from main.

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.

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.

Requirements:

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)

- [x] ~~Trivial iterator~~
- [x] ~~Input iterator~~
- [ ] Output iterator

ALGORITHMS (choose at least 1 from each category)

NON-MUTATING ALGORITHMS

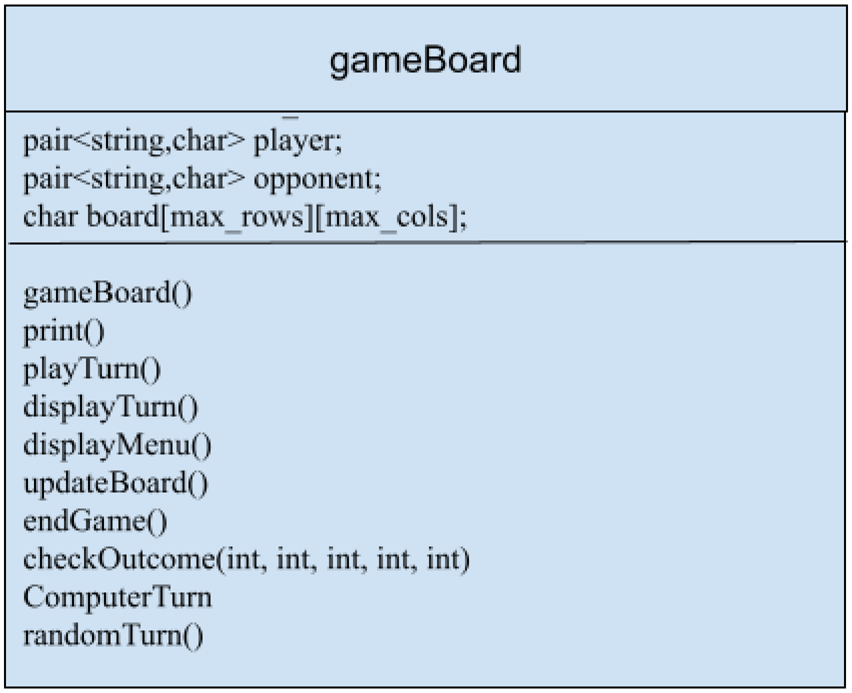
- [x] ~~Count~~

MUTATING ALGORITHM

- [x] ~~Swap~~
- [x] ~~Fill~~

Documentation of Code

Class UML

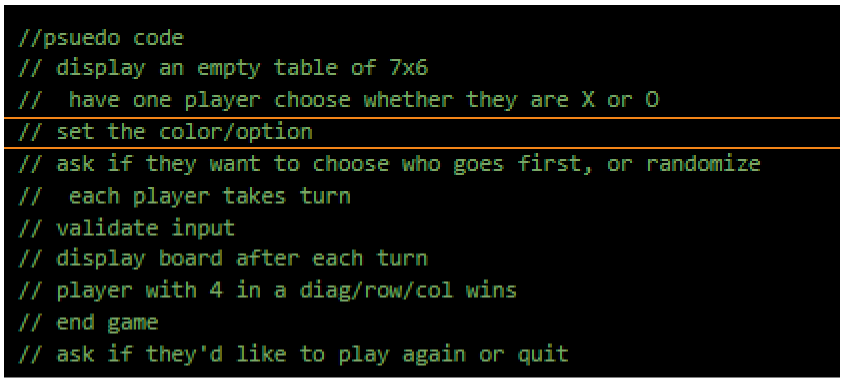


gameBoard

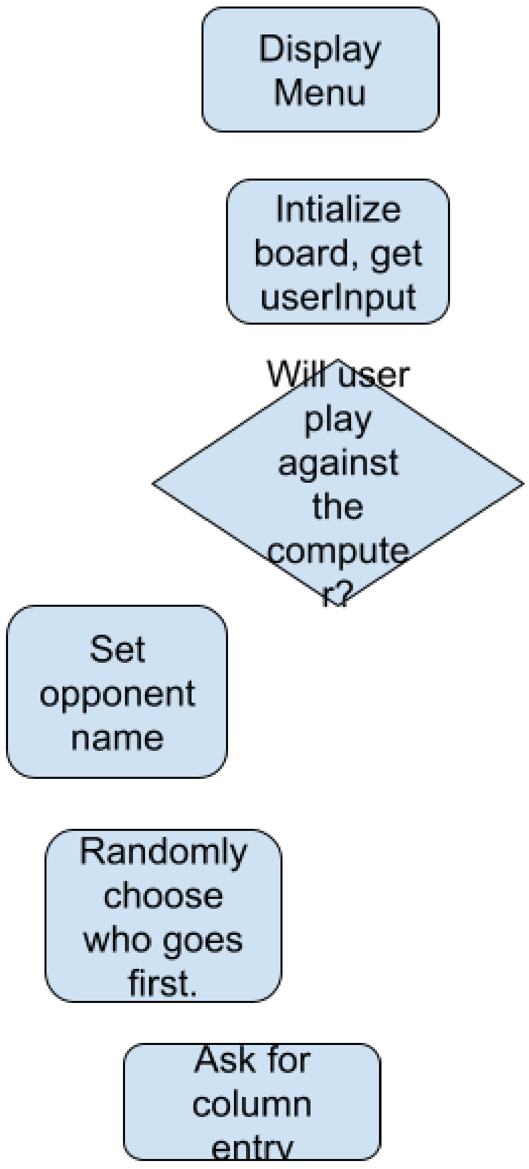
```
pair<string,char> player;
pair<string,char> opponent;
char board[max_rows][max_cols];
```

```
gameBoard()
print()
playTurn()
displayTurn()
displayMenu()
updateBoard()
endGame()
checkOutcome(int, int, int, int, int)
ComputerTurn
randomTurn()
```

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
```


Display Menu
Intialize board, get userInput
Will user play against the compute r?
Set opponent name
Randomly choose who goes first.
Ask for column entry