Project 2, CSCI 1730

Object Oriented, Command Line, Connect 4
Due date/time are on the Breakout Labs and
Projects webpage

Learning Objectives

- Design and implement a C++ project of moderate size, consisting of a main driver class and multiple class files and employing composition, inheritance and polymorphism.
- Use the "make" utility, a software engineering tool for managing and maintaining computer programs.
- Demonstrate knowledge of variable scope rules by predicting program output (i.e. testing your code since it will contain lots of scopes).
- Decomposing a complicated problem into simpler parts

Problem / Exercise

**You are encouraged to work with one partner on this project,
but you and your partner must both be registered for the same lab section
(note: only one submission on nike is needed for a group of two students).
Also, both students must contribute equally to their project.**

According to Wikipedia, `https://en.wikipedia.org/wiki/Connect_Four`, Connect Four is a two-player connection game in which the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to connect four of one's own discs of the same color next to each other vertically, horizontally, or diagonally before your opponent.

In our version of Connect 4, our players will use a char instead of colored disc. Player One will use the 'X' char and Player Two will use the 'O' char to simulate dropping colored discs down a column as shown in the examples on the labs and projects webpage. For this project, you will design and implement an object oriented, command line, Connect 4 game that uses C++ composition, inheritance, and polymorphism based on the requirements in this document and the I/O of your program must match the examples provided on the labs and projects webpage.

Project Requirements

Your program must adhere to all requirements stated below.

1. You will design and implement a Board class in two files: Board.h and Board.cpp. The Board class must have a default constructor, destructor, a printBoard() function, and any other members that you need to complete this project (yes, you need to design and implement the rest of this class as well as the other classes in this project). Your printBoard() function must print the board exactly as shown in the examples.
2. You will design and implement a Player class in two files: Player.h and Player.cpp. The Player class will serve as the parent class for HumanPlayer and SimpleComputerPlayer classes.
3. You will design and implement a HumanPlayer class in two files: HumanPlayer.h and HumanPlayer.cpp. The HumanPlayer class must be a subclass of Player. HumanPlayer objects should prompt a user for their name when its default constructor is called, and it should have function getName() that returns its name. Also, it should handle I/O from a human player as shown in the examples.

4. You will design and implement a SimpleComputerPlayer class in two files: SimpleComputerPlayer.h and SimpleComputerPlayer.cpp. The SimpleComputerPlayerPlayer class must be a subclass of Player. SimpleComputerPlayer objects always have the name of "Zoey", and their behavior is simple; they always choose the leftmost column that has an available space. There should be no pausing, no waiting on SimpleComputerPlayer to complete its turn, and it should complete its move without any human input (or input from the command line). It should choose the leftmost column that has an available space almost instantly (in a fraction of a second).

5. Your Player, HumanPlayer, and SimpleComputerPlayer clases must be polymorphic. Recall, a polymorphic class in C++ is a class that either declares (Player) or inherits (HumanPlayer or SimpleComputerPlayer) a virtual function.

6. `proj2.cpp` must contain the main function that processes command line arguments as shown in the examples, and it must use objects of the aforementioned classes to create a correctly working Connect 4 game that has four states:

   (a) Game in progress when no player has won and there is space available on the board for the next player's turn, and the game should move on to the next player's move after detecting this state.

   (b) Player One Connected Four and Wins! (program should end after detecting this state and printing this message)

   (c) Player Two Connected Four and Wins! (program should end after detecting this state and printing this message)

   (d) The board is full, it is a draw! (program should end after detecting this state and printing this message)

7. Your program must compile with all targets, dependencies, and commands given in the Makefile on the breakout labs and projects webpage for this project. This means that you must download, read, understand, and use the Makefile. When you are testing out your individual classes, you may want to compile and test each class separately, but your final version of your project should be work without any modifications to the provided Makefile.

8. All functions/methods should be commented properly.

9. You may assume valid input from the command line arguments and human players.

10. Your final program's executable, `proj2.out`, must have I/O that matches the examples exactly. This means that all of your classes and functions involved in I/O must follow these examples. Failure to follow the I/O provided in the examples may result in a failing grade for this project.

Examples

The I/O of your final program must match the examples on the labs and projects webpage (they are too large to show in this document).

1 Submission

Before the date/time stated on the Breakout Labs and Projects webpage, you need to submit your code on nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-proj2` if you are working alone. If you are working with a partner, then place all of your group work in a directory called
`LastName1-FirstName1-LastName2-FirstName2-proj2`. These directories must include all of the following.

1. All source files required for this project (proj2.cpp, Board.h, Board.cpp, Player.h, Player.cpp, HumanPlayer.h, HumanPlayer.cpp, SimpleComputerPlayer.h, SimpleComputerPlayer.cpp)

2. A copy of Makefile for this project

3. A README file filled out correctly (fill out a copy as you did in a previous lab, but remove all brainstorm sections since there are no brainstorms for this project). If you work with a partner, then put your first & last name and your partner's first & last name in the README file.

To submit from within the parent directory, execute the one of the following commands:

```
$ submit LastName-FirstName-proj2 cs1730
$ submit LastName1-FirstName1-LastName2-FirstName2-proj2 cs1730
```

If the submission is successful, then you'll see that a file that starts with `rec` (your receipt file) was created in the directory that you submitted. It is also a good idea to email a copy to yourself (to do this, modify the command found in Lab 01 for this lab).

2 Grading (50 points)

If your program does not compile on nike or a cf node with an unmodified copy of our Makefile, then you'll receive a grade of 0 on this assignment. Otherwise, your program will will be graded using the criteria below.

Makefile (working as aforementioned)	3 points
README file (filled out correctly)	2 points
All functions/methods are commented properly	5 points
Program runs correctly on various test cases on nike or a cf node	40 points
Late Penalty	-10 points for each 24 hour period late
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on nike or a cf node with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.