Connect 4 Project Specification

Mihir Kelkar, Vivek Alumootil

In this project we will program our own Connect 4 game. Connect 4, one of the most classic board games to play, has recently made its debut in the field of programming, allowing for various explorations in designing a computer player. Our Connect 4 game will follow the basic rules present in Connect 4, and introduce a computer vs player and player vs player aspect.

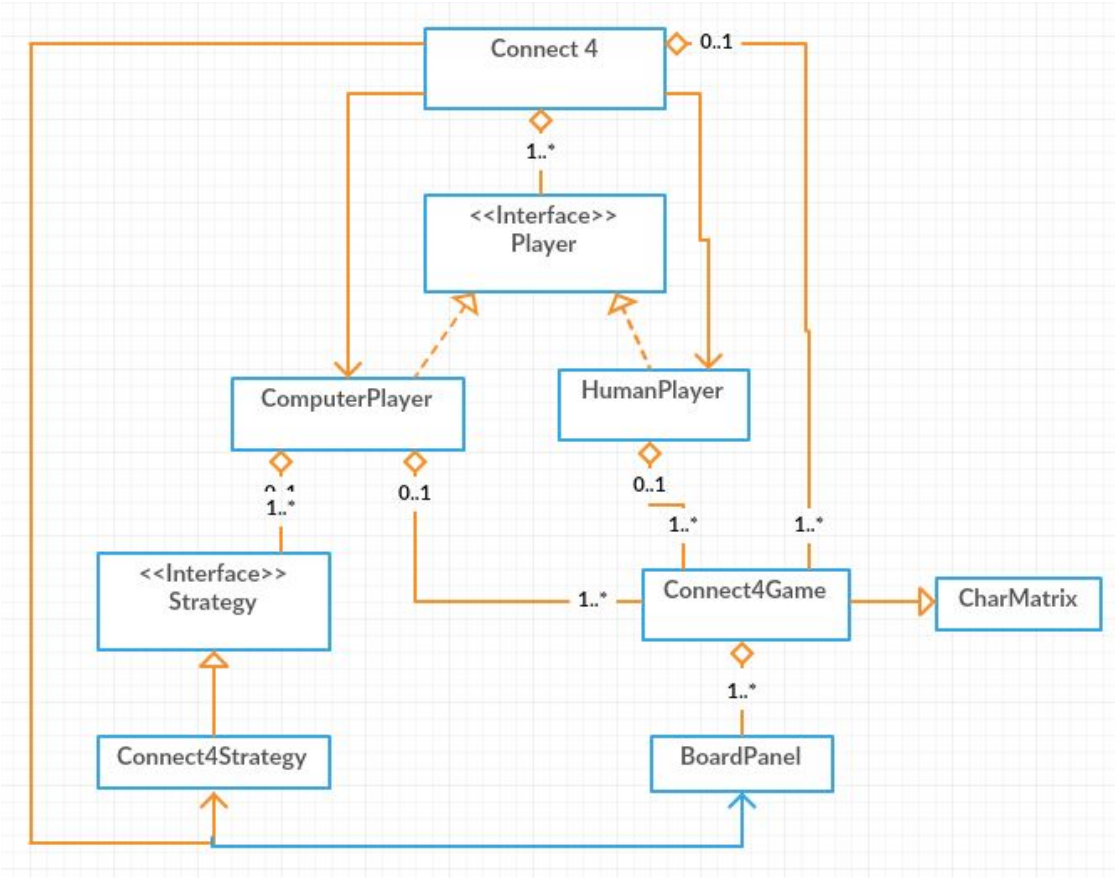
In Connect 4, an empty 6x7? grid is used to start. One player uses red checkers and the other uses yellow checkers. The players take turns adding checkers to the grid. Whichever column the player chooses, the checker will fall until it reaches a checker below it, or the bottom. Once a column is full, it can no longer be used as an option for the player's checker. The players alternate taking turns until there is a 4-in-a-row, or the entire grid is filled: a stalemate.

There are several challenges that come with this project. Some simpler aspects include determining the checker's position once it has been inserted into a specific column. Other, more challenging problems include checking for a 4-in-a-row. In order to do this, after every turn, we consider where the last piece was placed and check all the 4-in-a-row possibilities surrounding it, we could hard-code all connect-4 possibilities, and iterate through them to see if one matches. Another challenge is programming the computer player's defense and offense mechanisms, albeit defense is much easier than offense.

Our design for this project will consist of 3 parts: The structural design, the object-oriented design and the GUI. The structu

1. The structural design:
   This includes the data structures we will use for the project. We are using a 2D array to store the checkers in the game grid. An empty spot is represented by a 0 and a filled one is represented by a 1 for a yellow token and a 2 for a blue token. After a player does his turn, the computer replaces the 0 with the correct color code in the first available 0 in the column the player chose.
2. The object-oriented design:

(will create picture and do this later)



Our Classes:

The top class, *Connect4*, represents the main program window. Its constructor creates a *Connect4Game*, a *BoardPanel*, (the display panel for the board), and the human and computer "players." It also attaches the *Connect4Strategy* to the computer player.

A *Connect4Game* object models Chomp in an abstract way. It knows the rules of the game, and, with the help of its superclass *CharMatrix*, keeps track of the current board position and implements the players' moves. But *Connect4Game* does not display the board - that is left to a separate class: *BoardPanel*. *Connect4Game* only represents the "model" of the game. This class and the Strategy class are the only classes that "know" and use the rules of Connect 4.

The *Connect4Game* class extends *CheckerMatrix*, a general-purpose class that represents a 2-D of characters. A matrix of characters helps *Connect4Game* to represent the current configuration of the board. This class allows us to break the idea of the game down to basic programming structure, without all the GUI and other aspects of the game. This class would contain the more general functions such as checking for 4 in a row, and simulating the game environment (the checker falling from the top to the bottom).

The *BoardPanel* class is derived from the *JPanel* library, and is designed to display the current board position. *Connect4* adds the display to the main window's content pane and attaches it to *Connect4Game*. When the board changes (i.e a checker is dropped), *Connect4Game* calls *BoardPanel*'s *update* method to update the screen. This class' code has to deal with rendering the board animations, and changing the state of the board each time a play is executed.

The *HumanPlayer* and *ComputerPlayer* classes represent the two available players and both implement the interface *Player*. The *HumanPlayer*'s job is to take in the input of the column the player wishes to play in, and then execute that play properly. The *ComputerPlayer* class is designed to use the Strategy class to play its move. The strategy will be initially designed to play defensively. Coming with an offense will be a bonus for if we have time later.

Finally, the *Location* class represents a (row, col) location on the board. *Location* objects have basic methods to get adjacent locations, and access the row and col values of that specific location. In our game, locations will also have a state, such as yellow, red, or empty, to represent the checkers.

3. Detailed Design

The documentation will be provided for the final project specification.

4. Testing

Proper testing for this application will consist of making sure the overall game rules are accounted for, and the defensive strategy works as well as what to do for traps. The first set of tests will include exploiting the possible combinations of connect-4s to make sure the program has no errors with the basic rules. In this set, we will also test the case of a draw - the whole board is filled w/out a 4-in-a-row. The second set of tests will be focused on the defensive strategy employed by the computer player, and its ability to catch 3-in-a-rows and perform blocks as necessary.