FINAL PROJECT: Connect 4 Game

**CS111 Project Description**  
**Fall 2016**  
**Milestone 1**  
   
In this project you will be using the knowledge you gained from lectures and homeworks and applying them to a lengthier programming assignment. The answers to some of your questions do not strictly reside in this document. You are encouraged to look at other sources for inspiration. This semester’s project will involve the popular game Connect Four.  
   
**This is an individual project, and as such you will be held to the University’s academic integrity policies. You are welcome to discuss strategies and ideas with other students (and will be encouraged to in recitation), but you must code the project on your own.**  
   
The project will be divided into Weekly Milestones. You will have deliverables due every week as described by the milestone document. Below is a general overview of the project, as well as Milestone 1.  
   
This project is what you make of it.  There are points assigned to each milestone, which are further broken down into tasks. However, you should not be limited by what we suggest the minimum work required here is. To get the most out of this project, I urge you all to push yourselves to find your limits. You might surprise yourself! Good luck!  
Your assignment will be to implement a functional computer version of the game Connect Four. For those of you unfamiliar with the game and rules, please consult the Wikipedia article about the same: <https://en.wikipedia.org/wiki/Connect_Four>   
   
A basic introduction follows (copied verbatim from the Wikipedia Article): 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.  
   
**Milestone 1: Due 11/21 @ 11:59 pm. (5pm due time, grace period until 11:59pm) (Check, Check+, Check-)**  
Task 1: Play Connect Four  
This Milestone is the design phase of your project. However your first order of business is to actually play some Connect Four. You can pick up the physical game from many online and brick and mortar retailers and play with your fellow CS111 classmates, roommates, friends, family, etc. You’ll want to get a sense of the rules and strategies that make a good Connect Four player. Alternatively, you may visit one of the many websites that offer Connect Four as an online game.<http://www.pogo.com/games/connect4>  
   
Task 2: List information needed to play (submitted on sakai)  
After playing a good number of rounds of Connect Four, take a survey of the most basic information that one might need in order to program a game. Put this into a file called **design.txt**. Consider and note any error conditions that might occur from incorrect values of the inputs you’ve listed.  
   
Task 3: Design a basic game loop for playing Connect Four (submitted on sakai)  
Create a flowchart and/or pseudocode outlining the basic game loop. In other words, show the components of game play between two players and the correct ordering of operations. You need to be able to determine exit conditions, input parameters, etc. (Think back to the beginning of the semester when we were working with flowcharts, and what was necessary to be included). At this point, you should also consider how to organize parts of your flowchart into function units (think subroutines). The operations you may use in the flowchart are: storing values in variables, output (print), input, mathematical operations (+/\*-), looping, conditionals/decisions, comparators, String operations. Visit piazza.com if you have questions about any other operations. Please label this document **gameLoop**(.txt,.docx, .pdf, ...etc)  
   
Task 4: Optimizing Strategies for playing Connect Four (submitted on sakai)  
Now imagine you are building a hint system for a player of Connect Four. Your goal is to build an algorithm to suggest the next column a player should place their chip. Ideally, your algorithm will be better than your peers’ so that you can win. Sketch out your algorithms in any form you see fit.  Please label this document **strategy**(.txt,.docx, .pdf, ...etc)

***Milestone 2: Implement Board.java (submitted on autolab) 40 pts (Starting 11/21)***

* Implement a representation of the connect four board in a class called Board*

* Board representation details: You are free to implement any additional methods if you wish and include any data structures/variables you feel would be necessary. However, you MUST implement ALL of the methods described in Task2 for full credit.*

* (0,0) represents the upper left corner of the board. (i.e. if a player were to place a token in column 0, the first token to be placed would fall to (r-1, 0) where r is the number of rows in your board.*

* All positions in the board should be initialized to an empty space ‘ ‘.****Task1:****Define your fields (aka instance variables) necessary to implement the following operations****Task2:****Implement the following methods:*

***Your Board.java MUST have the following methods (DO NOT MODIFY ANY OF THE METHOD SIGNATURES)***

*Constructors:****Board();****//creates a default board of size 7 columns x 6 rows****Board(int row, int col);****//creates a board of size row x col*

*Both constructors above should initialize all positions to ‘ ‘****public int getNumRows();****//returns the number of rows in board****public int getNumCols();****//returns the number of cols in board****public char getPlayerOne();****//returns char representing player 1****public char getPlayerTwo();****//returns char representing player 2****public void setPlayerOne(char o);****//sets char representing player 1****public void setPlayerTwo(char t);****//sets char representing player 2****public char getToken(int row, int col);****//returns the char representing token at location row,col, returns '\0' if indices are invalid****public boolean canPlay();****//returns true if a token is still able to placed onto the board, false otherwise****public boolean play(int p, int c);****//places the appropriate token for player p in column c. returns true if successful, false otherwise.****public int isFinished();****//returns either the ID of the player who has won (1 or 2) OR 0 if the game has ended in a tie OR -1 if nobody has won yet*

*Milestone 3 HumanPlayer*

*2 Non penalized submissions allowed (1 pt deduction for every submission thereafter)*

*We have provided you a set of Java files to help you get started. As usual you should not change any class or method signatures we have given you. You may add methods to either of your implementations. Below is a description of the files and what your tasks are for MS3. While this MS is not totally dependent on a working Board.java (e.g. you can get a full score without a functional Board.java), it is much more difficult without one. You should ensure that your Board.java passes most of the testcases before you begin (we will open up submissions to test your Board.java after the deadline).  
  
Before you begin work, please read all of the instructions.*

1. *Player.java DO NOT MODIFY. This is the Interface that must be implemented by all Players. We did not discuss interfaces in class, but think of it as a compile time check to ensure that all players conform to a certain “interface”. In this case, all Players MUST implement the methods contained here in. The methods that must be implement by all types of players are:*

* + *playToken() – returns a column in which to play a token. This must be a valid column (e.g column must exist, and column must not be full)*

* + *lastMove(int c) – receives the most recent move from the other player*

* + *getPlayerID() – returns the player ID of this player*

* + *reset() – cleans up all state and allows the player to begin a new game*

1. *HumanPlayer.java This is the representation of a human player. The constructor takes parameters as described in the file. For the HumanPlayer, the methods should do the following:*

* + *playToken() – Asks the user for input, returns the column received from the user. In cases of error, re-ask the user for input.*

* + *lastMove(int c) – entirely up to you how you want to use this information (hint: you should keep track of where the other player has gone and where you have gone so that you can do adequate error checking in playToken())*

* + *getPlayerID() – returns the player ID of this player*

* + *reset() – clean up your state and prepare for a new game*

1. *ConnectFour.java – FOR YOUR USE ONLY (not graded). Here , we’ve given you the main game loop algorithm. Just fill it in with calls to your Player/Board objects. Create first with two instances of human players, and play the game against yourself/a friend. Ensure things seem to be working correctly. Test, Test, Test.*

*Scoring:****HumanPlayer.java****: Asks for user input, does error checking, always plays valid token.*

Milestone 4:  
  
In this final milestone, you will be implementing a GUI interface for your game. The source code you submit will be via sakai, as the GUI will be human verified.  
  
Using the ConnectFour.java game driver, edit the code to make use of the GUI code we have given you. Descriptions of the classes and their relevant methods appear below. Also general tips on what to include in your driver also appear below.  
   
ChipColor.java – an enumerated type that allows you to specify a token’s color. Usage of this class allows you to specify colors. *e.g.*ChipColor.BLUE specifies the color blue, while ChipColor.RED specifies the color red. Use as parameters to the CFGUI constructor  
CFGUI.java – defines the main GUI of the board. The constructor takes the three parameters – the game board (of type Board), player 1’s chip color (of type ChipColor [ see above]), player 2’s chip color – in that order  
               Method Summary

* CFGUI(Board b, ChipColor c1, ChipColor c2) – constructor
* void redraw() – method must be called every time your board is updated
* void gameOver(int player) – method must be called when your game is over and winner is determined. Parameter is player’s id. Pass in 0 in the case of tie.
* void close() – method must be called before exiting your game (closes the GUI window)

GIO.java – allows you to output and input from the GUI. Your use is optional. You may modify HumanPlayer.java to use GIO.readInt() if you wish, or continue to use IO for inputs.  
               Method Summary

* static int read…(String prompt) – ask the user for a … using the prompt given as a parameter in a pop up window. Returns said …. Where … is any of int, boolean or String
* static void displayMessage(String message) – displays the given string as a pop up message in the GUI

Hints and tips:  
               In your ConnectFour.java file, you must:  
                              Create a CFGUI object  
                              Call redraw() everytime there’s a change on your board  
                              Call gameOver() to notify the GUI of who won  
               You have complete freedom in how you want to implement your game. You can modify any of the files we’ve given you (but must submit them!).  
               Ideas for implementation: custom chooseable colors, Human v human, human v computer easy, human v computer hard, etc. This is your time to be creative. If you’re doing anything that would require the player to do anything special, please  make a notation in a file called readme.txt  
What to submit:  
               Every thing you need to make your program compile and run.  
Scoring:  
               10 points if we are able to play your game (without errors)            
               0 points for anything else.

***Code for part 4:***

public class ConnectFour{  
      
      
    public static void main(String[] args){  
          
        //Create new board object  
        Board game = new Board(6,7);  
        CFGUI set = new CFGUI(game, ChipColor.BLACK, ChipColor.RED);  
        Player p1 = new HumanPlayer(1,6,7);  
        Player p2 = new HumanPlayer(2,6,7);  
        game.setPlayerOne('o');  
        game.setPlayerTwo('x');  
          
        while(game.isFinished()==-1){  
            //player 1  
            if(game.canPlay()==true){  
                int i=p1.playToken();  
                  
                while(game.play(1, i)==false){  
                    i=p1.playToken();  
                }  
                System.out.println("P1 made their move");  
                p2.lastMove(i);  
                set.redraw();  
                if(game.isFinished()!=-1){  
                    break;  
                }  
            }  
            //player 2   
            if(game.canPlay()==true){  
                int i=p2.playToken();  
                  
                while(game.play(2, i)==false){  
                    i=p2.playToken();  
                }  
                System.out.println("P2 made their move");  
                p1.lastMove(i);  
                set.redraw();  
            }  
        }  
        //player 1 has won  
        if(game.isFinished()==1){  
            set.gameOver(1);  
            set.close();  
        }  
        //player 2 has won  
        if(game.isFinished()==2){  
            set.gameOver(2);  
            set.close();  
        }  
        //It's a draw  
        if(game.isFinished()==0){  
            set.gameOver(0);  
            set.close();  
        }      
    }  
}