Connectivia

Brenna Carver, Caylene Parrish, and Stephanie Frankian

Final Project

CS 230

Phase III

**Connectivia**

Our project involves the popular game Connect Four, with a slight twist. In order to place the token on your turn in the place you choose, you must correctly answer a question within a specific category. You would specify which category of questions you would like to answer at the beginning of a match. If you answer correctly, the token is placed where you would like to place it. If you answer incorrectly, you will lose a life. The game is won when a player achieves four (or possibly more) tokens vertically, horizontally, or diagonally, or when a player reaches zero lives.

**1) User’s Manual:**

Upon invoking the program, the game specifications portion of the Connectivia Game will run. The GUI that results will prompt the player to choose the difficulty level of the game. A Hash Table would be generated with questions as the keys, and with answers as the values, from the text file associated with the category level. By generating a Hash Table following the player’s difficulty choice, one is able to easily add new questions and answers to the text files so that the game can be continuously modified and updated with new questions going forwards.

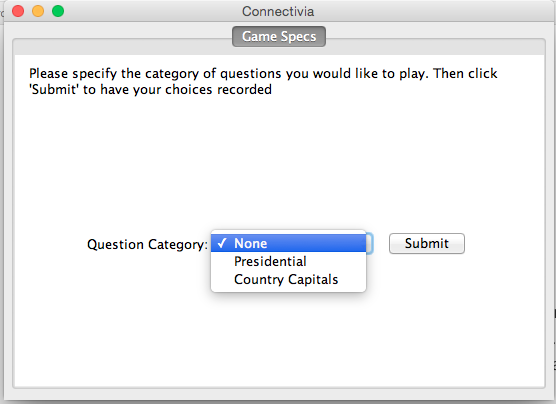
Upon submitting these choices, the GUI containing the ‘board game’, questions, game history, etc. will be invoked. This second GUI interface will be relatively straightforward. There will be three separate tabs, “Connectivia”, “Instructions”, and “Match History”. “Connectivia” contains the Connectivia board game and the question dialog box, and would be the landing page upon the invocation of the GUI. The player whose turn it is can choose a location on the Connectivia board to place a token via the six buttons above each column. Each token falls to the lowest position in the board (because of gravity, of course!). Upon clicking a column in which to place a token, a dialog box will pop up with a question and with four possible answers. Only one answer is correct, and answering correctly results in the placement of the token, while an incorrect answer results in the decrementing of a life.

The “Instructions” tab contains the instructions for how to play the game. Connect-Four usually is a game where a player drops a token into a specific slot without interference. Because we are introducing the question barrier and the total lives component, the user may need to read our modified rules to understand how to play and win the game.

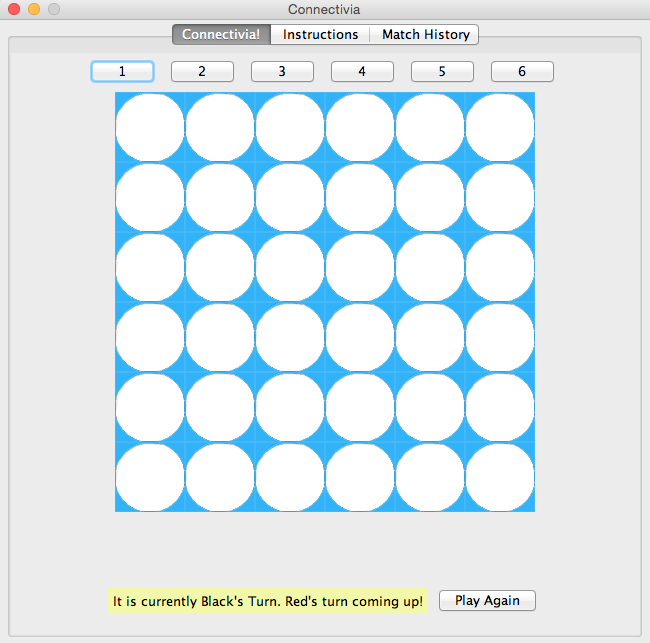
The “Match History” tab will record the wins for both players in the current invocation of the GUI. This will only be updated after a game ends.

The game will end if a player runs out of lives—resulting in the victory of the other player—or if a player achieves ‘Connect-four’, or four tokens that are aligned horizontally, vertically, or diagonally.

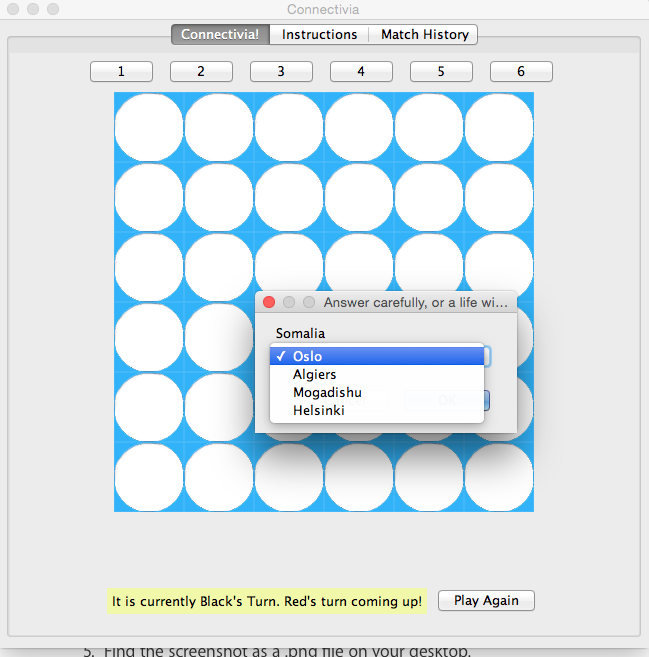
1. Initial GUI



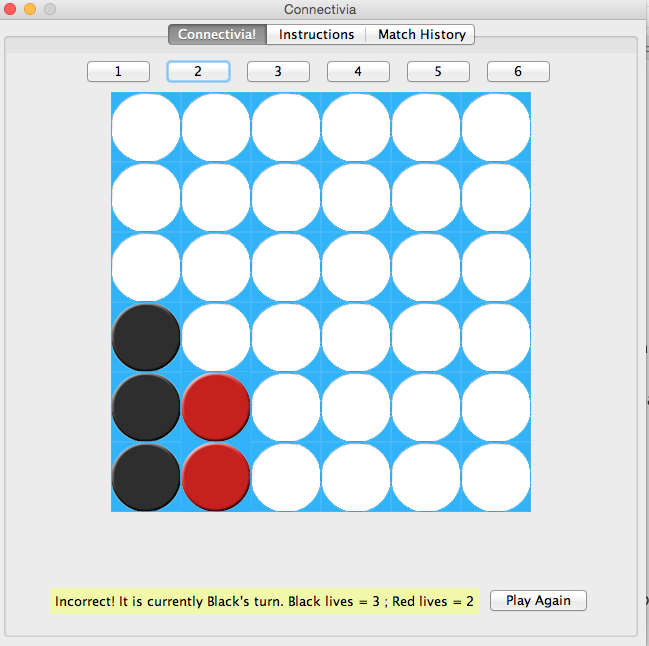
2) Landing Page, Connectivia!, with its initial blank board. This is also how the panel looks when Play Again is clicked.



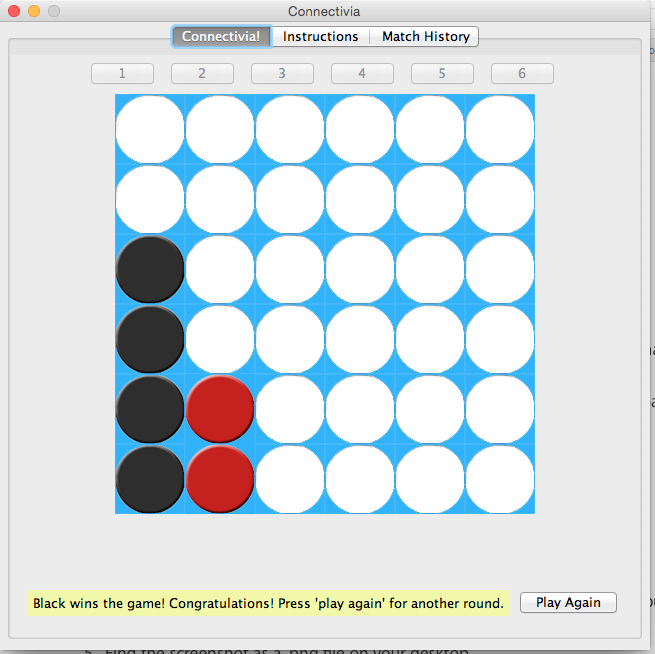
3) After clicking on one of the column numbers, the question dialogue box pops up.



5) An instance of a game in session, which also shows the lives being decremented with an incorrect answer.



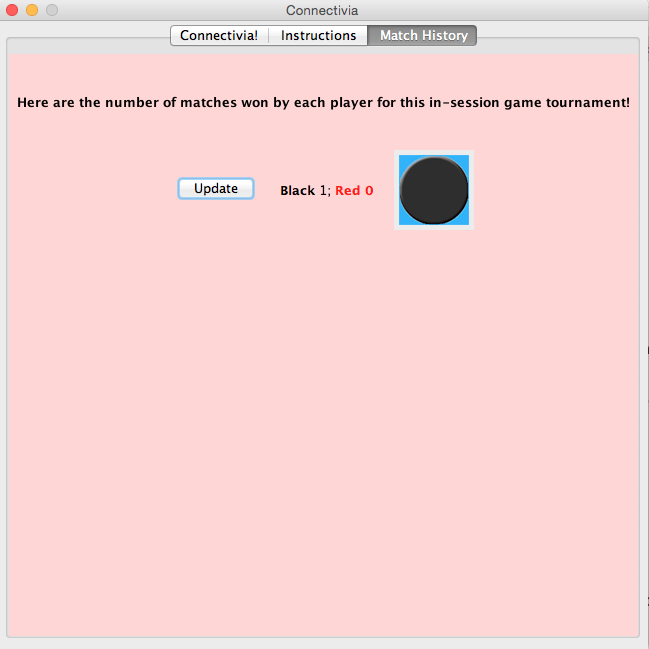
6) A game that was won by ConnectFour vertically. The buttons are disabled.



7) The Instructions panel.



8) The Match History Panel, that counts the total wins for both players.



**2) Technical Report:**

**a) ADTs**

There are two collections of data we will need to store questions and their corresponding answers, and the players’ wins and losses.

ADT 1: Hash Tables

For the questions and answers, we will use hash tables. Hash Tables are ideal because they can store corresponding keys and value, which will represent the questions and answers. The Hash Tables will be created by reading in text files.

ADT 2: LinkedList

For the players’ wins and losses, we will utilize LinkedLists, one for Player 1 and one for Player 2. Each player’s LinkedList will be initialized as empty, and after each game, a 0 will be added to the losing player’s LinkedList, and a 1 will be added to the winning player’s LinkedList. To count the players’ total number of wins, we will be able to loop through and add up all the 1s. In this case, LinkedLists are desirable because of the simplicity and linearity of the data we will be storing, as well as the ease of adding to them.

**b) The Important Classes**

The initial GUI class will have methods that will generate the game screen based on the user selections of category. The other GUI class will display the main game screen and will contain methods that handle user clicks on the submit button as well as clicks on the Connect Four Board when appropriate.

*GUI-relevant classes:*

1) **class GameSpecs**

The class GameSpecs creates the an initial GUI to determine if the user wants to play with which category of questions they prefer. A new GUI window is created upon clicking ‘Submit’ (via a button) of the input information.

2) **class ConnectiviaGUI**

The class ConnectiviaGUIis generated via the category choice in GameSpecs and serves to create the next GUI, which contains the following three Panels as three tabs on the same GUI:

3) **class BoardPanel**

The class BoardPanel creates one of the panels that is added to the GUI. Specifically, this is the panel that contains the gameboard, the question field, and a status of the current player. This is also the landing page for the GUI generated by the GameSpecs.

4)**class AboutPanel**

The class AboutPanel contains the instructions for playing the game, and exists on the BoardPanel panel.

5) **class MatchHistoryPanel**

The class MatchHistoryPanel returns the total wins of the two players in the current function invocation.

6) **class Total**

The Total class will create an instance of GameSpecs to get the user-entered difficulty information, which will generate a GameDriverGUI containing the three aforementioned panels to display the game board and respond to the user’s actions.

*Functionality-related classes:*

7) **class QuestionsAndAnswers**

Reads in a file, creates and tracks the questions that are to be answered by the players, and records the questions answered correctly by each player separately.

Actions:

The constructor accepts a file path input for a text file and creates the Hashtable with the appropriate values.

**public String[] generateAnswers(String question)**

The method generateAnswers() returns an array of Strings based on the given question. The array is randomly populated with the 3 incorrect answers to other questions and the one correct answer.

**public String generateRandomQ()**

The method generateRandomQ() generates a random number and returns a random question in the HashTable.

**public boolean isCorrect(String question, String answer)**

The method isCorrect(String q, String a) checks if the given answer matches the value in the HashTable question/answer key/value pair.

**8) class Board**

The class Board controls the Connect Four aspect of Connectivia. Through the use of a 2D array, the class handles game tokens placed into its board and determines whether there was a winner or not.

Methods:

The Getters get and return the corresponding value.

**public int getWinner()**

**public int getCurrentTurn()**

**public int whoseTurn()**

Setter:

**public void changeTurn()**

The setter changeTurn() changes the player whose turn it is.

**public int findColumnLow(int column)**

The method findColumnLow(int column) takes a int value that represents a column number. It then searches for the first open spot in the column and returns the position of this open slot.

**public void placeToken(int column)**

This method takes an integer value as a parameter and places the token of the current player into the board based on this value. The integer is retrieved based on which column button the user presses in the GUI.

**public boolean checkTie()**

The method checkTie() loops through the 2d array representing the connect four board to make sure there was not a tie. A game would be tied if every single token was placed and no one had four in a row.

**public boolean checkColumnVictory()**

The method checkColumnVictory() loops through the entire gameboard and returns true if it finds any instance where there are 4 tokens of the same kind in a column. It also updates the instance variable, winner, to indicate if player One or player Two had the winning 4 in a column move.

**public boolean checkRowVictory()**

The method checkRowVictory() loops through the entire gameboard and returns true if it finds any instance where there are 4 tokens of the same kind in a row. It also updates the instance variable, winner, to indicate if player One or player Two had the winning 4 in a row move.

**public boolean checkDiagonalVictory()**

There are two diagonal scenarios in Connect Four. checkDiagonalVictory() looks to see if either one of these types of diagonals are present in the board. If any of these scenarios are satisfied, then true is returned and the method ends. Otherwise, we check each scenario and if there is no diagonal victory, false is returned.

**public boolean isGameOver()**

The method isGameOver() returns true if the game was won in any of the 3 different ways or if a tie occurred.

**9) class Game**

The class Game simulates a game of Connectivia by mostly drawing on Board’s determinations to smoothly check the progress of the match.

Methods:

**public int determineWinner()**

The method determineWinner() checks for victory by either the opponent’s loss of lives or by achieving connect-four on the game board.

**public boolean askQuestion()**

The method askQuestion() generates a dialog box with a question for the user as well as four answers to choose from. The method returns true if the user answers correctly, false if incorrectly. A life is decremented from the answering player if false, and prevents the placement of the token in the chosen area.