Assignment 2 Documentation

Hassaan Malik 1224997

Katrine Rachitsky 1306314

Trevor Rae 1324949

Navleen Singh 1302228

Paul Warnick 1300963

**Requirements**

1. Enables the user to set up a 7-by-6 Connect Four frame
2. The frame should include two different coloured discs (blue and red), with one on each side of the frame
3. At the start of the game, no discs should be entered in the frame
4. The order of play should be determined randomly
5. The user will be able to choose to start a new game
6. When all discs the user wants to place have been placed in the frame, the system should analyze whether the current state is possible or not (if there is no disc beneath a disc, it can only be supported if it’s position is at the bottom of the board), and all errors will be highlighted on the screen (including an unbalanced number of blue discs in comparison with red discs)
7. The final state of this frame cannot include any winning connected “line” of 4 discs of the same colour
8. Allow the players to be able to make legal moves
9. Recognize when the game has been won or if the game cannot be won
10. Results of the game must be displayed at all times
11. User must be able to save and load a game

**Design**

5.1

The first class is called Model. This class is responsible for linking all the methods; it initializes the action listener and the game panels. It calls all of the classes and passes the appropriate data into them so that they may execute with the appropriate arguments.

Another class, Disk, constructs an abstract data type “Disk”. This class allows us to create objects of type Disk that are made up of two integer values: x and y. These are the x and y coordinates of each disk object, which are used to represent the location of each disk on the game board when a game is being played.

Another class used in the design of the Connect4 game is View. This class is responsible for creating the buttons, the labels and the background images. This class essentially sets up the images for the graphical user interface, setting up each panel in the game (the game board, the start screen) when they are called.

Lastly, we implemented a class “Control” that is responsible for obtaining the location of each mouse click and dealing with it appropriately. In this class, we implemented an action listener that, depending on the panel the program is currently in, executes the corresponding action.

5.1

We chose to decompose the project into these classes to prevent coupling while allowing for maximum cohesion; with only four classes, each class is mostly dependent on its own methods, gathering minimal data from other classes and therefore avoiding problematic dependencies by reducing dependencies on other classes.

**Classes and Modules**

Model Class

* 5.2 Public Methods
  + Model(): a no argument constructor that allows a static method to call a non-static variable or non-static method from the Model class
  + Model(JPanel game\_panel, boolean developermode): initializes game panel, calls developer
  + mouseCicked, mouseEntered, mouseExited, mousePressed: all take in MouseEvent e and are required, override methods used to allow mouse clicks
  + mouseReleasted(MouseEvent e): when the mouse is released, uses the mouse function method in control
  + actionPerformed(ActionEvent e): is a button listener, which calls the button function in control
  + main(): calls appropriate methods in order to create the main window, and displays the panel
* 5.2 References to Requirements
  + satisfies requirement 5, allows user to enter game screen when they click button “Start Game”
* 5.4 Private Methods & Variables
  + createFrame(JFrame main\_frame): sets up the location of the window as well its size
  + The variable dev\_mode is used to check if the game needs to be run in developer mode or not. This variable is then used again in the control class when the user clicks the developer mode button.
  + The variable card\_layout determines how the panels are shown. This variable is used in view to create what it looks like and control to display it
  + The variable deck\_panel is the main panel in which the card layouts are displayed. This variable is used in view to add the panels to the deck and control to display it when the user clicks the appropriate button.
  + The variable main\_frame makes a new JFrame type named "Connect Four" (This will be the name of the window) this variable is used in view to create the size and control to display it when the user clicks.
  + The variable check\_disk checks the disk for if there is a piece on the position that the user has clicked. This is used in control when the user clicks a spot on the board
  + The rest of variables in this class are private and are not used by any other class. These variables are used to start the main program.

Disk

* 5.2 Public Methods
  + Disk(): default constructor that prevents an error from occurring if disk is called without any arguments
  + Disk(int x, int y): constructor to set the x and y coordinates for a Disk object
  + setX(int x): sets the value of the x coordinate in a Disk object
  + setY(int y): sets the value of the y coordinate in a Disk object
  + getX(): returns the x coordinate of the Disk object
  + getY(): returns the y coordinate of the Disk object
* 5.2 References to Requirements
  + an abstract data type that allows game pieces to be created, contributing to satisfying requirements 6 and 7
* 5.4 Private Methods & Variables
  + no private methods were used in this class
  + Private variables included in this class are the x and y coordinates of the Disk object. These variables are part of the abstract data type built for each game piece; it is beneficial to keep them private to preserve encapsulation. The coordinates that each Disk object holds indicates the position of a game piece. These variables are maintained in this class because the user will never see the coordinates of their pieces, and they need not know about the implementation since they simply need to place them.

View

* 5.2 Public/No Modifier Methods
  + titleScreen(): creates title screen, sets up the locations of the buttons
  + infoScreen(): displays the instruction screen, loads the buttons for this screen
  + gameScreen(): displays the game board, as well as 2 disk pieces; this method also loads the buttons for the game board
  + developerScreen(): creates the buttons for the developer screen and sets up the screen for developer mode
* 5.2 References to Requirements
  + this class satisfies requirements 1, 2, 3 and 10 by setting up the game board and displaying disks and displaying the player win results.

• 5.4 Private Methods & Variables

* + the variable game\_panel is being used in the class control, this variable holds the current game panel and control takes this panel and displays it when the mouse is clicked
  + a boolean variable developer\_mode for checking if the user wants to use developer mode or not. This variable is used to recognize if developer mode is in progress or not.
  + The rest of the variables in this program are private and are not used by any other class. These variables are used to load all the pictures from the images library, set the game window designs, place the locations of the buttons etc.

Control

* 5.2 Public Methods
  + buttonFunction(ActionEvent e, JPanel panel, boolean dev\_mode): contains all functions of each button. When a button is clicked, this method finds the name of the button in order to decide which action should be executed (for example, if exit is clicked, a confirmation popup will appear to confirm whether or not the user wishes to close)
  + mouseFunction(MouseEvent e, JPanel panel, boolean dev\_model): this method is called when the mouse is clicked. It checks if that click is within the range of the position it’s clicked it then sets the x and y values of the position it clicked it. It then checks if dev\_model(developer mode) is on or off, if it is on then it places the disk in the position of the board that is clicked. If dev\_model is turned off then it places a disk at next available position of the column.
* 5.2 References to Requirements
  + This class satisfies requirements 1, 4, 6, 7, 8, 9, and 11 by allowing a random player to go first, allowing the user to place a disk on the game board. It allows the players to be able to make legal moves, recognizes if the game can be won or not won and allows user to load and save the game.
  + In developer mode, this class allows the game board to place disks on the game board.

5.4 Private Methods / Entities

* + blueDisk(int x, int y, JPanel panel,boolean gamewin): sets the location of the blue disk, as well as the panel it is located on. If the game has been won, it will change the pieces to a blue win piece to indicate where the blue person won
  + redDisk(int x, int y, JPanel panel,boolean gamewin): sets the location of the red disk, as well as the panel it is located on. If the game has been won, it will change the pieces to a redwin piece to indicate where the red person won.
  + playerNameSet(JPanel panel): prompts the user to enter each player’s name, displays the name in the proper locations of the game board, this method also checks if the names are in the correct format such as if they entered in no name, if two names are the same etc and prompts the user if it is correct or not.
  + nameOnScreen(JPanel panel): displays the names of each player on the panel in the correct position.
  + pieceAir(): determines whether or not there is an empty space underneath the location the user tries to put a disk. If there is empty space (if there is either a disk underneath or the piece has reached the bottom of the board), this method returns true; otherwise, the method returns false. This methods uses two for loops to run through every position on the board except for the bottom row, then using an if statement it checks if a blue piece of value 1 and a red piece of value -1 have a black space of value 0 below them.
  + win(): checks whether or not 4 pieces have been placed in a row vertically, horizontally or diagonally; if so, the method returns true.Otherwise, the method returns false. The method runs within two while loops for running through each column and row, a simple counter is used to keep track of how many pieces are currently being connected in all positions around the board. After each piece is placed this counter is incremented and checked to see if that pieces result in a win. Once the counter reaches 4 a win message is display tell who has won, and the game is over
  + noMoreMoves(): checks if the game running has no more moves left, by moving through the board after each move is made and checking whether there is enough open room for a win to be made. To explain each time a player (blue or red) places a piece an algorithm will run with a number of “for” loops checking if there is at least enough space to make a win with this piece in particular. The loops check all directions around all pieces, for example, every piece in the board has a check for four spaces to the left, right, and both diagonal and vertical directions. Once the whole board has been checked, if all of the pieces have no possible win from it then the game is a draw. This is done for both player (i.e. if one player can no longer win the other may still have the chance (a default win))
  + saveGame(String savename, String player1,String player2): using a printwriter this method saves the player names, and the positions of all the pieces, then saves them into a save folder. It runs through the board using two while loops and records the number of each piece placed at each position. If it hits a blue piece it saves a -1 or -1 for red and if it blank it stores a 0.
  + loadGame(BufferedReader read): using a BufferedReader this method loads the game from the save file. It scans through the file for the player names, and the locations of each of the pieces in a while loop. Once the method is done running it returns a double array with the locations of all the piece.
  + The variables in this class are private because it deals with the mouse clicks, which only need to be dealt with within the class; the values need to be maintained so that accurate calculations can be carried out with them. If these values were to be altered in any way, methods that use user input could be compromised. For this reason, we keep these variables private

# 5.3 User Relations

Note: Arrows pointing from one module to another means the module that is being pointed to is being used by the module that is pointing to it.

**4.6 Internal Review**

The design we used for the Connect4 game closely resembles the MVC pattern. The Model creates the window of the game, while View provides the user interface that is created by Model is used to display the game to the user through panels. The Control handles the mouse and button clicks, which serve as input. This component deals with the user’s interaction with the game.

**Test Report:**

We tested by running the classes and testing their collective effectiveness for various functions:

* To test buttons, we would run the program and click on the button we are trying to test. If it performed the desired function (going to a new game screen/panel, starting the game, etc.), the test passed.
* To test if the players names are correct, click start game, enter types combinations (for example starting with a space, only a space, ending with a space, containing numbers, one name not being filled in, both names not being filled in, etc.) of names, if the name is in an incorrect format a window will pop up showing an error message, the test has passed
* To test loading a game, click the load game button and check for all combinations of loading files, clicking the cancel button and loading the game as many times as possible to see if the same result occurred.
* to test the standard game board, we would run the program, click the Start Game button to access the game board, and:
  + Click a position on the Connect4 board. If a disk was placed in the position clicked, the test passed
  + Click a position in the middle of the game board (so that the position is more than one position above the bottom of the game board and/or a disk). If the disk does not “float” (is not placed in a position with empty space directly beneath), the test passed
  + Click on positions within the window but outside of the Connect4 board. If disks were not placed outside of the board (or within), the test passed
  + Click two positions, one after the other. If both positions have been filled with a disk, and each disk is a different colour, the test passed
  + Use the mouse to place four disks of the same colour in a row, if a pop up window with the message indicating the proper player has won AND the four pieces that were connected are now highlighted with black circles, the test has passed
  + Use the mouse to place pieces in random spots around the board, keep track of the pieces until they are moderately close to the top (a draw scenario is possible). Check after every place if a winning combination is possible for either player. If there are no winning combinations then available a window will pop up telling the user this, the test has passed
  + To test saving a game, click the save game button as many times as possible to see if an error would occur,, clicking the cancel button and saving the game as many times as possible to see if the same result occurred.
* To test miscellaneous functions of the standard game board, follow the same process to reach the standard game then:
  + Make note of the who is prompted to move first in the pop up window. If the correct players name is is highlighted in black, the test passed
  + Alternate moves between players using the mouse, if each time the user clicks the proper players name is highlighted, the test has passed
  + Click on the Main Menu while on the game screen, if a warning message pops up telling you to save if you haven’t already, the test passed
  + Click on the Instructions button (while on game screen). Once at the instructions screen click on main menu. If a warning to save window pops up and no errors occur when the user cancels, the test passed
* to test the game board in developer mode, we would run the program, click the Developer Mode button to access the game board for developer mode, and:
  + Click a position on the Connect 4 board. If a disk was placed in the position clicked, the test passed
  + Click a position in the middle of the game board (so that the position is more than one position above the bottom of the game board and/or a disk). If the disk “floats” (is placed in whatever position was clicked, regardless of whether or not the position below is empty), the test passed
  + Do the above, and then click Start Game. If the user is prompted with an error, the test passed
  + Click on positions within the window but outside of the Connect4 board. If disks were not placed outside of the board (or within), the test passed
  + Click on four positions in a row, with the same coloured disk, then click Start Game. If the user is prompted with an error, the test passed
  + Click Select Red, and place a piece on the board. Click Select Blue, and place a piece on the board. If the first click resulted in a red disk placed on the first position clicked, and the second click resulted in a blue disk placed on the second position clicked, the test passed
  + Click on 2 positions with one coloured disk, and 1 position with the other colour. If the user is prompted with an error after clicking Start Game, the test passed