Phillip Pavlich 001414960
Prakhar Jalan 001450321
Dinesh Balakrishnan 001409123

2AA4 ASSIGNMENT 1 DOCUMENTATION

4.1.

A description of the classes/modules you have decided to use in your application, and your explanation of why you have decomposed the application into those classes

Description of classes/modules:

There are four classes used in our application.

- moveLocations (1)
- PlayerPiece (2)
- SixMensMorris (3)
- StdDraw (4)

(1) This object class is used to be able to map out the positions on the board to the pieces and then determine whether these spots are occupied or not. The primary characteristics of the moveLocations object are the " box coordinates" of the clickable location (minX, maxX, minY, maxY) and its occupancy.

(2) This object class is used in order for each piece to have its own individual characteristics initially defined and modified throughout the game. By having a color and a piece number, we can accurately identify the specific piece of each player, as well as modify its position as the game goes along.

(3) This main class is responsible for creating the user interface (calling methods from StdDraw), creating the 16 moveLocation objects, the 12 PlayerPiece objects, and combining everything into a runnable program. The main method calls a variety of branching methods such as newGame() in order to maintain an organized set of code.

(4) This module is used in order to gain access to drawing tools for the game's user interface. Several methods within this library were used to design the main menu screen as well as the game board and its pieces.

Decomposition Explanation:

By decomposing the application into these 3 classes (StdDraw given module), we organize the program in a clear and understandable manner. Specifically, the two object classes are made to retrieve individual data regarding the 16 possible movable positions and the 12 pieces used within the game. This makes it easy to identify and modify the state of the game by manipulating these objects accordingly. Additionally, it is evident that the main class (SixMensMorris) is used to organize the information needed in the game and sequentially develop the game's state. For example, the drawing of the board is done separate from the button functions themselves (newGame). The continueGame function is structured quite differently than the newGame, as it allows the user to place as many pieces as they require, and informs them of any errors. By separating the classes in this specific way, we easily create all the objects needed into the newGame method and continueGame method, store them in their own arrays for later use, and then read the user's mouse clicks corresponding to the locations they wish to place their pieces. There are certainly other ways of approaching this application, but we chose to go with our intuition and attack the problem with our initial concept of the class layout.

4.2.

For each class, a description of the interface (public entities), and make sure that there is a description of the semantics (behaviour) of each public method in the class, as well as a description of the syntax;

- moveLocations (1)
    - Constructor (1.1)
    - getMethods (1.2)
    - checkOccupied and changeOccupied (1.3)
- PlayerPiece (2)
    - Constructor (2.1)
    - getMethods (2.2)
    - changePiecePosition (2.3)
- SixMensMorris (3)
    - Main method (3.1)
    - mouseInRange (3.2)

    - mainMenu(3.3)
    - setBoard (3.4)
    - newGame (3.5)
    - continueGame (3.6)
- StdDraw ~ Given module

(1.1) This constructor initializes and creates the moveLocation object based on the parameters given: minX maxX minY maxY. The occupancy is also initialized to be false when the object is created.
(1.2) The 4 getMethods are the accessor methods for each bound of the clickable location (minX maxX minY maxY), where the object's corresponding characteristic value is returned.
(1.3) These occupancy methods are for returning boolean values (true or false) regarding whether the spot itself is occupied or not (checkOccupied) and modifying its occupancy (changeOccupied).
(2.1) This constructor initializes and creates the PlayerPiece object based on the parameters given: color and piece number. Also, the piece's activation is initially set to false and the piece's position is set to 0 (not placed).
(2.2) The 3 getMethods are the accessor methods for each of the primary characteristics of the PlayerPiece object. Its color, piece number, and piece position are returned when these getMethods are called (respectively).
(2.3) This method takes in a new position as a parameter, and modifies the piece's current position to this new value. This allows the piece to essentially "move" upon the board as its position is needed as a characteristic (backend).
(3.1) Since this is the main method, everything here is directly related to the user interface and the basic sequential structure of the program. We start of by calling the mainMenu method to build a mainMenu screen using StdDraw. Next, we check whether the use is clicking on a button or not, and if so, take them to that corresponding screen. The newGame method allows players to place their pieces upon the board accordingly, while the continueGame method allows them to set up their previous board, and inform them of any setup errors thereafter.

(3.2) This method returns true only if the mouse is within clickable bounds of a given location. The method takes in the mouse location (x,y) and compares it to the given (minX maxX minY maxY) values to determine whether the mouse is in range.
(3.3) This uses StdDraw to draw the entire main menu. The methods are used directly from StdDraw to create the two appropriate button pictures, and other visuals. This method is purely for visuals.
(3.4) This uses StdDraw to draw the game board. The methods are used directly from StdDraw to create the two appropriate board, player piecesn and other visuals. This method is purely for visuals.
(3.5) This method starts off with one of the player's going first (randomizer), and then alternates their turns to place their pieces upon the board. Here, the PlayerPiece objects and moveLocation objects are created, and then every case of clicking (position placement) is checked; once the user places their piece, the program modifies specific values to shift the state of the game appropriately.
(3.6) This method starts off with the user being able to place blue or red pieces upon the board (to their liking), and inform them of any possible errors (once they click the done button). To swap colors, we created two buttons (blue and red), so that they can switch at anytime. In the method itself, the placement occurs within a while loop that breaks once the user clicks the done button, and then displays text concerning any possible errors.

4.3.
A view of the uses relationship;

There are many uses of relationship in this application. Everything relates back to our SixMensMorris class. A simple relationship can be seen by the interaction of the startup screen where there is the option to go to "Start New Game" or "Continue New Game". If new game is clicked, t will start a new game from scratch while they can start a game where they can play all the pieces beforehand to start a game and they will also have an option to check if there is any error, if they decide to click on continue game. SixMensMorris uses Std draw to set up the graphical representation of the screen and the game board. It uses the StdDraw class to draw out the startup screen, the game board with all the

lines and points, the pieces and even the text. The moveLocation class relates back to SixMensMorris as it first creates the 16 positions available for the pieces to be clicked on by the mouse. Then, it will also make sure only one piece will go and occupy the position and not let the rest of the colours go and will stop from future pieces to get that spot as it will be occupied. This class also lastly, make sure there will be less available spots and the combination of available spots will be one less for the next team. The last relationship will be one between the PlayerPiece class and the SixMensMorris. Here, they interact as they take the player pieces from this class as the playerpiece class contains specific information of each piece such as the colour, number, and position. The SixMensMorris class then will take this information and collect all 12 pieces. It also chooses who will go first and who will go second.

4.4.
Include a trace back to requirements in each class interface;

The application was made sure to have all the requirements through the four classes. The moveLocations class made sure to deal with the requirement of being able to move all the pieces to 16 possible positions, but less if there are positions occupied which is needed too. This makes sure the pieces does not also go wherever it wants, and limited to correct positions, such as not overlapping pieces or out of the game board.

The PlayerPiece class generates all 12 pieces and stores each attributes. This meets the requirement of holding all the specific information of each pieces. It can hold the number of pieces, piece's colour and the position of the pieces which is important to track and keep for the application.

The StdDraw is an important class that allows this game to work as it is the basis of the overall application. This class allows us to meet the requirement of the graphical representation of the board by providing us with the capability to create drawings in our program. Through a simple graphics model, we are able to create the lines, points, circles, and rectangles that are all used in our application. Lastly, our main class is the most important one.

The SixMensMorris class contains the basis for the application. It starts up with the introduction screen allowing the user to start new game or continue game. It draws the graphical representation of the board. This also class also

takes care of the requirement of choosing who gets to go first. The order of play will be chosen randomly over here. This also makes the available 16 position clickable so the pieces can be moved and it also checks if it will be occupied so the pieces can be moved to. It will take care till all the pieces are placed.

4.5.

For each class, a description of the implementation (private entities), including class variables - include enough detail to show how the class variables are maintained by the methods in the class;

For the moveLocations class, there are 5 private entities. There is the double minX, double MaxX, double minY, double maxY and lastly the boolean occupied.
These private entities are maintained through the accessor methods within the class through getMethods. The first 4 methods return the current object's corresponding values, while the 5th method (occupied) maintains the object's position occupancy upon the board.

For the PlayerPiece class, the private entities are the String pieceColor, int pieceNum, boolean pieceActivated, int pieceRadius, and the double piecePosition. These 5 entities are implemented and maintained through the accessor method within the class with getMethods. The pieceColor, pieceNumber, and pieceRadius are private entities that will remain unchanged, as we only need to access this information to identify individual PlayerPiece objects. The piecePosition and pieceActivated however will change as the game progresses, since these need to be modified based on the user's decisions of piece movement/placement. They are easily maintained through the use of the getMethods, as the piecePosition can be modified when given a new position value, and the pieceActivation is initiallized as false due to being prior to placement.

4.6.
An internal review/evaluation of your design.

We started the design process by brainstorming how to implement an interactive game like Six Men's Morris. We came to an agreement that StdDraw would provide a nice colourful design that looks appealing in which we can still implement an interface for the user. We wanted to modularise the code so that it is broken up into tiny bits of code. When these modules are put together, it creates our game. Creating classes for pieces and locations on the game board were a crucial step in our game design. These allowed us to keep track of all the pieces as they are moving around the board. We are also able to check if a location is occupied in order to prevent a player from putting a piece on top of another piece.

Get methods for the pieces and locations came in handy for allowing the user to click on locations or pieces that they wish to place or use. We kept some variables inside our classes private to implement separation of concerns. We knew that it was important to keep parts of the code unknown to the user to protect our implementation. I think our implementation of our design went a long way to making assignment 2 and 3 easier. We removed potential problems and made our code have a strong structure to it.

Our design also consists of a continued game where the user can place as many pieces for both sides as he wants in locations. If the user sets up the board in a way that is invalid, error messages are displayed. The two possible errors are if the user placed more than 6 pieces for one side and if less than 3 pieces on one side were placed. If less than 3 on one side, it is a sign that the game is over or that the players have not finished placing their men on the board.

4.7.
Document the code so that it is clear how the code follows its design, and also explain design decisions in the code that were not included in the design document.

To document the code, we have put some comments into our code for each class and method. Before each method is a description of what the method

does. We have explained all the design process and decisions on how to implement this game in the previous sections, especially 4.6.

5.

Include a test report document that records how you tested your application (we have not discussed testing yet – so you are on your own with this document

We tested our design and code using numerous strategies and criteria. First off, when the code is run there is no syntax errors meaning that we have a functional code. We designed our code with StdDraw so that we can provide the user with an attractive program that also uses interactive features. StdDraw does not have buttons so we used methods from that class to make our own buttons. We made it check if the mouse was clicked and the x and y component were both situated inside the button. To test this, we ran the code and tried to click the button and it continued us to the next step in the game. When we did not click the button it keeps going until you do just like it is supposed to.

We tried to play the game, in order to check if the player's turns alternate as they set up the board with their pieces before they start moving the pieces. We also did a test to click the start new game button to make sure the board is setup correctly on the window. The board needed to have straight lines connected for the two squares and little dots and all possible locations that you can put a piece. One aspect of this game that is not legal is to place a piece on a location that is already occupied by another piece. To test this, we tried the setup of our game and clicked on an already occupied location. The game will not allow you to make a move there as it is invalid. We tested all the get methods and mutators in our game pieces class as well as our board location class. We printed out the values after calling these methods to ensure we got the expected output.

We tested the randomizing of which player goes first. We played several games to ensure that there were some games that Blue was able to go first and others where Red was able to go first. We also tested our ability to overwrite already drawn elements on the window. To get rid of an element, we made the assumption that you can draw something over top of it to hide the element. Our

hypothesis was correct as we were able to draw rectangles over top of unwanted elements to make that element disappear. Our continue game button is a button that allows the user to put pieces where he would like to set up them up and then click done. Once he clicks done, it will output any errors in the board setup. The possible errors are that one player doesn't have 3 players on the board meaning the game is won or the setup hasn't finished. The other error is that more than 6 pieces were placed on one side. We have tested the errors to make sure they are printed out for those two cases. Another factor that we considered when testing was the positioning of the elements. It is important that all circles are centered on the location so that it doesn't mess up the implementation of the game. To test this, we have clicked on all possible piece locations to ensure that every piece that is placed on the game board lands on the proper location.

Testing is one of the most important parts of coding in order to ensure correctness and efficiency. We have made sure to do all the necessary tests to prove that our code creates a user friendly interactive setup to the game Six Men's Morris and allow users to place their pieces on the game board.