Software Engineering Assignment #3

John Peng
Jason Balaci
Haipeng Li
Jingxuan Li

Objective/Requirements:

The objective of our project is to implement a graphical interface for the game 6 Man Morris. The implementation achieves the following goals:

1. Implements a graphical object representing the game board, which allows players to put discs of colours red and blue onto its different positions
2. The different positions occupied by the discs (ie. the state of the board) must be some valid combination allowed by the rules of the game, and all errors are reported

The first person to go is randomly determined at the start of the game.

Architectural Design:

We decided to go with the MVC design architecture for our implementation of 6 Men Morris. The advantages of using this design model are:

Separation of concerns:
Divides the parts of the application into different components that are not closely related. This allows our group members to work on individual modules with minimal communication and coordination between our members. Our development time has been significantly improved due to our ability to work concurrently on different parts of the project, as well as reducing the overhead of communicating between group members.

Modularity:
Thinking about an application as one entity can sometimes be difficult, especially for complex problems. Modularization allows us to break it down into easier to design smaller pieces, and then assemble it together in the end again.

Modules

Modules in our application falls into one of the 3 components of the MVC design architecture:

Model:

Deals with data representations, and holds the current state of the game. Modules belonging to this category include:

Disc.java (enum)

- an enum class object that can take on the values RED, BLUE, and EMPTY
- Used to represent a disc to be placed on the board; empty board positions are also occupied by an EMPTY disc

- **Traceback to requirement:**
- Public enum Disc{ } : allow player to place a RED or BLUE color disc.
  Called by ControllerImpl.java to select color.

Depth.java (enum)

- an enum class object with values INNER and OUTER
- used to represent the inner and outer loop of the board
- used in combination with Direction.java to represent a unique position on the board

- **Traceback to requirement:**
- Public enum Depth{ } : Defining the INNER and OUTER parts of board.
  Called by BoardModel.java to form the INNER and OUTER parts.
  Called by ControllerImpl.java to move disc bwteen INNER and OUTER parts.

State.java (enum)

- an enum class object with values PLACE_DISC, REMOVE_ENEMY_DISC, MOVE_DISC and END
- used to represent the current state of the game; certain actions can only be performed by players at certain stages of the game (ie. if the game is in the PLACE_DISC state, the player can choose to place a disc at an empty location on the board; this action can only be performed during this state of the game)

- **Traceback to requirement:**
- Public enum State{ } : Used to represent states.Called by ControllerImpl.java in order to change the game state.

Direction.java (enum)

- used as a compass on the board. Each disc can be said to be either NORTH, NE, E, etc of the middle of the board. Direction is an enum that helps with determining whether the Disc is N, NE, E, SE, etc through angles off of an imaginary axis created in the middle of the board, with origin in the middle of the board.
- used in combination with Depth.java to represent a unique position on the board
-

`public boolean isAdjacentTo(Direction dir)`	Check if the direction is adjacent to this one through math, except for EASTERLY directions

`Public Direction[] adjacentDirections()`	Check if the adjacent directions were cached,iterate over all needed possible directions,cache the found adjacent directions and return it.
`public double toAngle()`	return the angle off the pos x-axis (in dir of the y-axis)
`public boolean isPole()`	return whether the model.Direction is dominant/a pole

- **Traceback to requirement:**
- Public Direction[ ] adjacentDirections( ) : To check directions.

BoardModel.java (class)

- the main model that holds the current state of the board, as well as providing methods for accessing and modifying its states
- BOARD is a HashMap <Depth, HashMap<Direction,Disc>> that holds the Disc state(RED, BLUE, or EMPTY) of every position on the board determined by Depth and Direction
- when BOARD is first initialized, the value of the Disc object at every single position (Depth,Direction) on the board is set to EMPTY

Method (Access Program)	**Description**
`public BoardModel()`	constructor
`public boolean isLocEmpty(Depth depth, Direction dir)`	returns true if the Disc state at position (depth,dir) is EMPTY; false otherwise
`public Disc getDiscAtLoc(Depth depth, Direction dir)`	returns the Disc object at position (depth,dir)
`public void setDiscAt(Depth depth, Direction dir, Disc disc)`	changes the Disc object at position (depth, dir) to the Disc object disc
`public void removeDiscAt(Depth depth, Direction dir)`	changes the Disc object at position (depth,dir) to Disc object EMPTY
`public boolean isDiscInMill(Depth depth, Direction dir)`	checks if the Disc at position (depth,dir) is in a Mill
`public boolean isValidMove(Depth fromDepth,`	Check if the Disc is making a valid move

`Direction fromDir, Depth toDepth, Direction toDir)`	from it's current position to another position
`public void reset()`	sets the Disc object at every position on the board to be EMPTY

- **Traceback to requirement :**
- Public boolean isDiscInMill (Depth depth, Direction dir) : For checking Mills
  Called by ControllerImpl.java to check the game state (Mills) and return.
- Public boolean isValidMove(...) : For checking valid moves.
  Called by Main method while every move is confirmed and return.
- Public void setDiscAt / removeDiscAt (...) : For Placing or removing disc.
  Called by ControllerImpl.java to place or remove disc.

GameModel.java (class)

- Using two private variable to set the color for each player
- Disc currentActor indicates the colour of the current actor
- Constructor of the board,can reset or init the default data
- Has the functionality to swap the players
- Added the boolean variable versusAI in order to set the game model of pve or pvp.

Method (Access Program)	**Description**
public void swapActor()	Swap the current actor with the other actor
public void reset()	Clear the board initialize the default value
public BoardModel getBoardModel() /public void setBoardModel(BoardModel boardModel)	Get and set the boardmodel
public Disc getActor1Colour/ public Disc getActor2Colour public void setActor1Colour(Disc actor2Colour/ public void setActor2Colour(Disc actor2Colour	Get and set the actor's color
public int getActor1DiscsLeft()/ public int getActor2DiscsLeft() public void setActor1DiscsLeft(int actor1DiscsLeft)/ public void setActor2DiscsLeft(int actor2DiscsLeft)	Get and set the value for actor that how many discs can still be placed

public int getActor1DiscsLost() / public int getActor2DiscsLost()	Get and set the value for actor that how many

Controller:

Contains the logic for updating the view/model, based on the current state of the application.

Controller.java (interface) & ControllerImpl.java (class)

- Controller contains the method definitions while ControllerImpl actually implements them
- ControllerImpl implements Controller and DiscClickListener
- this is the main controller class for the project
- GameController will update the state (State.java) of the game based on user inputs
- GameController is initialized with the following instance variables:

ControllerImpl's Instance Variables	**Description**
`Private Members`	
`private final Gson gson`	Gson object used to saving/loading the game
`Private bot bot`	The AI object
`private GameModel gameModel`	GameModel object used to contain all information about the current game
`Private boolean versusAI`	Implement the versusAI.The boolean variable versusAI is using to decide the game model,pve if returns true otherwise pvp .
`private Depth move_lastClickDepth;`	for move phase, the origin of the disc
`private Direction move_lastClickDirection;`	for move phase, the origin of the disc
`private final BoardView boardview`	BoardView.java object

- ControllerImpl has the following methods (inherited from either DiscClickListener, Controller, or declared by itself):

Method (Access programs)	Description
Public Methods	
`public GameController()`	constructor
`public void placeDiscAt(Depth depth, Direction dir, Disc disc)`	performs the PLACE_DISC state where the currentActor places a Disc at a particular location
`public void removeEnemyDiscAt(Depth depth, Direction dir)`	performs the REMOVE_ENEMY_DISC state where the currentActor gets to choose one of the enemy's discs and remove it from play
`public void moveDiscTo(Depth fromDepth, Direction fromDir, Depth toDepth, Direction toDir)`	performs the MOVE_DISC state where the currentActor gets to move one of his/her discs along the board
`public void restart()pvp`	Restarts the game in the case of pvp
`Public void restart()pve`	Restarts the game in the case of pve
`public void saveGameTo(File file)`	saves the game to a file
`Public void update`	Implement boardview,update the imformation from jframe
`public void loadGameFrom(File file)`	loads the game from a file
`public Disc getCurrentActor()`	returns the coloured disc of the currentActor
`public void onClick(Depth depth, Direction direction)`	implements DiscClickListener's onClick method, it is used to update the board, perform actions, and does nothing if no action is necessary.
`public GameModel getGameModel()`	returns the game model
`Public getWinner()`	Return winner's disc color
Private Methods	
`private boolean hasAnActorLost()`	returns true if one of the two actors have lost at 4 discs
`private void lossByTrap()`	called when a player loses by being trapped in (leaving them no discs that can be moved). It sets the state to END and notifies the game of the winner

`private boolean canOpponentMakeAnyMoves()`	checks if the opposing player can make any valid moves, returns true if the opponent can

- **Traceback to requirement:**
- The ControllerImpl.java is the highest 'commender' to other components. It contains Main method.

Bot.java

- Implement a bot as a opponent for pve option.
- it is coded for AI, given it functionality of placing discs,making valid moves and removing the Discs under the rules of the game.

Bot's Instance Variables	**Description**
private final Thread thread	Import the java.thread in order to make a thread of execution in the run method .
private final Random r	Import the random r,which can be used to place the disc in the random location.

Method(Access Program)	**Description**
public void start()	Starts the thread if true returns
public void run()	1.check if it is under the condition of a pve game 2.check if the location is empty run the case of place the disc 3.check the validation before each move makes then move the disc from the current location to the target location 4.run the case of removing the disc if the target location has the same color variable with the player.

DiscClickListener.java (interface)

- this is a basic interface with a single method inside of it (onClick, with signature (Disc, Direction)), no default implementation
- intended to be used whenever the user clicks a particular position on the BoardPanel's rendered board display

Main.java (class)

- just contains the main(String[] args) that all Java programs require to run.
- solely used to create an instance of GameController.java and let GameController do the rest

View:

Creates a Window for the user to interact with through Swing and AWT.

BoardView.java (class)

- this controls the entire view/graphical user interface of the program
- has two important components; creating a window and updating the information displayed on said window
- creates an instance of BoardPanel to actually render the board

Method(Access Program)	**Description**
`public BoardView()`	constructor
`public BoardPanel getBoardPanel()`	returns the instance of the BoardPanel currently being displayed on the UI
`public void update()`	updates the information displayed on the ui
`Private void forceRedraw`	force the boardPanel to be repainted immediately

- **Traceback to requirement :**
- Displaying the board(backgroud).

BoardPanel.java (class)

- extends JPanel so that it can have similar behaviour and be treated as a Component
- override's JPanel's paintComponent method to implement it's own
    - draws the board
    - draws coloured discs where the player placed them
- contains a basic onClick disc system so that any other Controller may create an implementation of the onClick method from DiscClickListener.java and have it called whenever a disc location is clicked

Method(Access Program)	Description
`public BoardPanel()`	constructor
`public void paintComponent(Graphics g)`	overrides JPanel's paintComponent method with intention of drawing a Six Men's Morris board
`private void drawPoints(Graphics g)`	renders the coloured discs on the board
`private void drawBoxes(Graphics g)`	renders the boxes of the board
`private void drawDots(Graphics g)`	renders the small black dots on the boxes of the board to show the user where the valid click locations are
`private Point toPoint(Depth depth, Direction direction)`	converts a Disc location to a particular Point (x,y) on the JPanel
`public void placeDiscAt(Depth depth, Direction direction, Disc colour)`	places a coloured Disc at a particular location on the board
`public void removeDiscAt(Depth depth, Direction direction)`	removes a coloured Disc from a particular location on the board
`public void mouseClicked(MouseEvent e)`	implement's MouseListener's mouseClicked method with intention of implementing an onClickEvent-like system. It will call all registered onClick (DiscClickListeners) using the anchored disc click location on the board if applicable
`private boolean inrangeof(double x, double target, double epsilon)`	returns true if x is within epsilon distance of target
`public void registerDiscClickListener(DiscClickListener listener)`	registers a DiscClickListener for the EventBus-like system discussed in the description for mouseClicked
`public void reset ()`	Remove all the discs from the board.

- **Traceback to requirement :**
- The method to finish all the steps input by players.
  Call by ControllerImpl.java.

Internal Review:

After reviewing our final implementation with the team, we have decided we have followed the MVC design principles perfectly and therefore has no room for improvements (at least when it come to the software design process). We took full advantage of the object oriented design paradigm to design modules that encompassed important design aspects such as modularity, separation of concerns, and encapsulation. Our modules were developed independently and concurrently, and required minimal effort and coordination to put together. Thus, it is not a statement of arrogance to say that we cannot find any improvements to be made to our code, but rather, it is a confidence in the fact that we followed the MVC design principle to the best of our ability and wrote some quality code. Our program runs on Linux, Windows, Mac, and it should work on any operating system with a working Java 8 JVM implementation.

Testing:

Throughout the development of our application, we performed white box testing and tested each component individually to ensure that when it is coupled with another module, that it would not present any issues or errors. After we performed white box testing, we performed black box testing by running the application and placing a few discs randomly. White box testing each individual module was the more strenuous method, however, it was the most time effective and it proved itself to be the better of the two methods for a small program because when we performed black box testing, we found no errors.

Test	Expectation	Result	Pass/Fail
Place disc at all locations	Screen update with new disc	<-	Pass
Create 3 in a row (Mill)	Move to REMOVE_ENEMY_DISC phase	<-	Pass
Both players place all 6 discs	^	<-	Pass
Move disc phase	If there is a new Mill, then it should go to REMOVE_ENEMY_DISC phase, otherwise, continue the game	<-	Pass
Trap all enemy discs	Trapped player loses	<-	Pass
Remove 4 of enemy	Player who captured 4	<-	Pass

discs	enemy discs wins		
Attempt to move disc	Disc should move to adjacent disc loc	<-	Pass
Prev and Post click required for ^	<-	<-	Pass
Make sure all moves, placements, and mills are valid	<-	<-	Pass

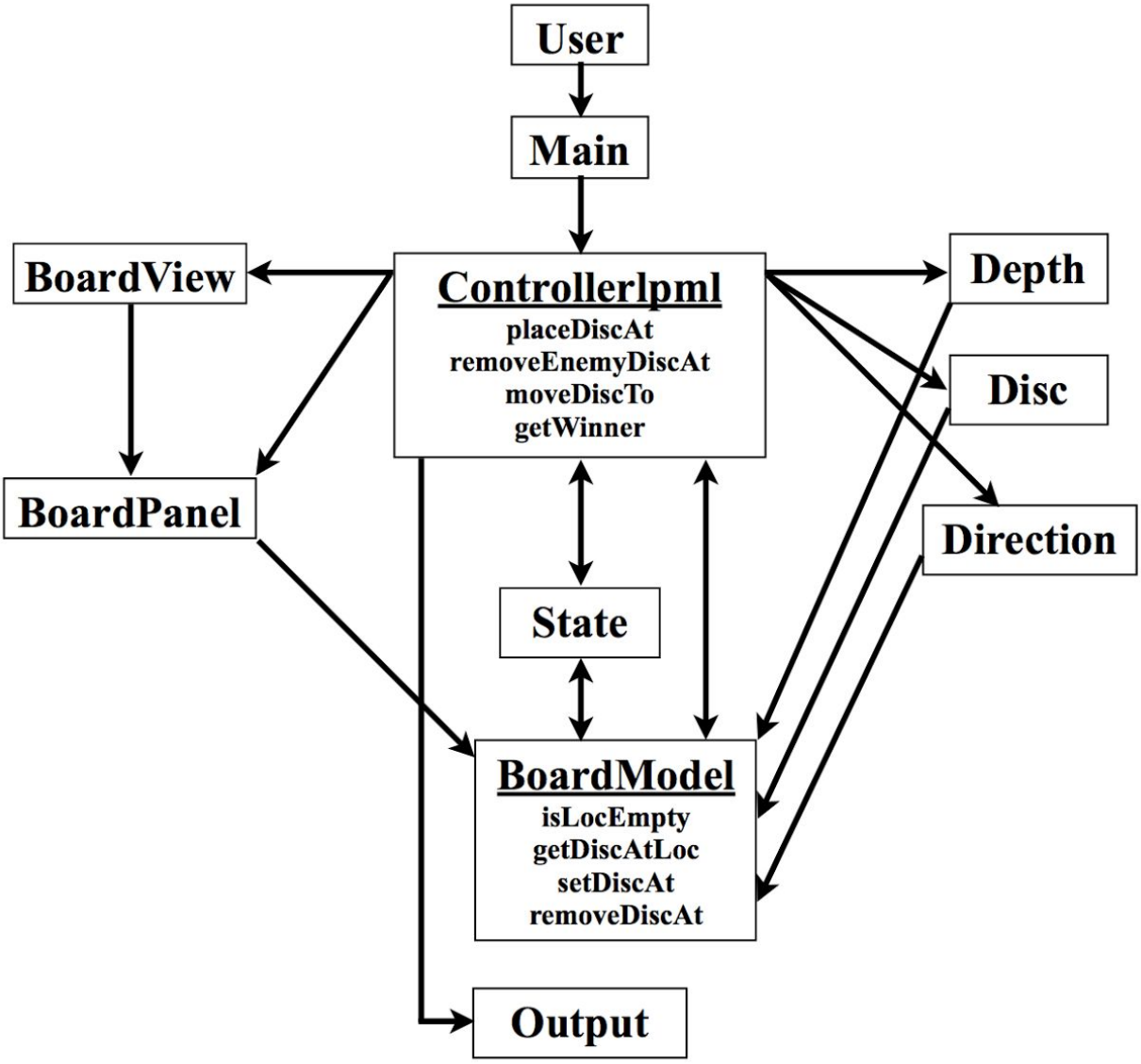
Traceability:

The Main.java is the main method required by java. The ControllerImpl.java is worked as Main method for this program, in order to call other methods (BoardView.java, BoardPanel.java, Depth.java,etc..) and contain the main functions, then it will calculate result(Blue wins, Red wins, Game draw, Game in progress) by calling other boolean methods (isDiscInMills,etc..) and return to the interface. BoardView.java and BoardPanel.java together to generate the game board (Interface) and immediately give feedback according to the inputs (Players movements), Depth.java and Direction.java will locate the Disc, Disc.java will make Blue disc and Red disc. BoardModel.java will store outputs returned by other methods(Depth, Direction, etc..). The program is threaded through the usage of Java Swing but there are still only two main entry points for inputs, the program main method and mouse clicks on the gui (which are handled by the BoardPanel, BoardView, and ControllerImpl).

Module	Uses	Trace Back
Main.java	Main method	N/A
BoardModel.java	Store data	N/A
Depth.java	Give the depth	Able to move disc
Direction.java	Give the direction	Able to move disc
Disc.java	Display the disc	N/A
GameModel.java	Rules	N/A

State.java	Check the state	Give the result
BoardPanel.java	Work with BoardView	Draw background
BoardView.java	Work with BoardPanel	Draw background
DiscClickListener.java	Check the player move	Enable player to move
Bot.java	Computer AI	Add AI
Controller.java	Control	N/A
Controllermpl.java	Control implementation	Enable player to control

Uses Relationship:


User
Main
BoardView
ControllerIpml
placeDiscAt
removeEnemyDiscAt
moveDiscTo
getWinner
Depth
Disc
Direction
BoardPanel
State
BoardModel
isLocEmpty
getDiscAtLoc
setDiscAt
removeDiscAt
Output