6 Man's Morris: 2AA4/2ME3 Assignment 3

Gregory Smilski, 1404091
Abigail Gaulin, 1327924
Karl Knopf 1437217

April 8, 2016

Contents

1 Introduction

This document describes the java project 6 Men's Morris.

1.1 Architecture

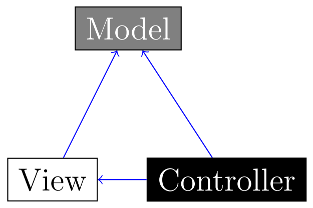

Figure 1: MVC Model

This software uses MVC architecture in its design. MVC stands for Model, View, Controller, and is a design tool used in software development. The View contains all information the user sees, and interacts with the user. The Model contains all the data, and the Controller contains commands which modify the view and model. This architectural style is useful as it allows for modulation, and parts of the program can be modified without affecting any others.

1.2 Technologies

- Java: An object oriented programming language
- Eclipse: It is an IDE used for the development and testing of software typically in Java
- Java Swing: A java toolkit designed to aid programmers in the creation of gui applications. This widget toolkit allows the programmer quick access to various predefined graphical objects, allowing the easy creation of a graphical interface.

2 Modular Decomposition

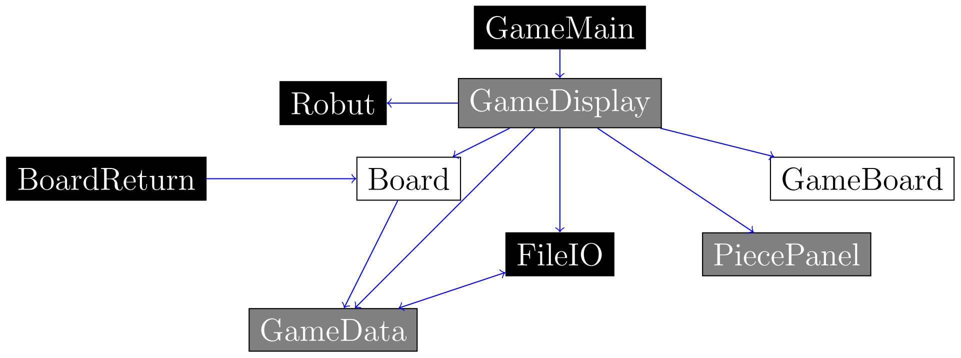

Figure 2: Modular Decomposition Outline

$\rightarrow$ *represents that the module uses the module the arrow is pointing to*

The modules chosen (as detailed in the Module Guide), were done so in order to make it straightforward for the 3 members involved in the project to code individually, and also to enforce modularity in the program. The GameDisplay, GameBoard and PiecePanel make up the View. The PiecePanel creates the tokens to take the user input, GameDisplay creates buttons to check or restart the game, and GameBoard displays the game data to the user. By dividing the View up in this way, if any of these functionalities need to be modified that can be done without affecting the other parts.
new
Board ,as well as part of GameDisplay make up the Controller. This controls the game logic, and is called by the View to make modifications to the game data based on user input. It makes sense for the Controller to be separate from the other modules, as any changes that are made to the game logic can be implemented without there being any change to what the user sees/interacts with, and likewise any changes to the user interface will not interfere with the game logic/data. The class robut is also part of the controller. It manages the logic of the AI.
The model is contained in the GameData class. Here the game information

is stored, and it can be used by the view and the controller. The controller is also able to update the model.

3 Module Guide

3.1 MIS

- GameMain.java
  This module houses the main function of the program, and allows it to run. It creates a game display object the projects to the screen.
  Interface
  Uses
  - GameDisplay.java
  - javax.swing.SwingUtilities;

  Return Type
  - None

  Access Programs
  - public main(String args[]) : A main method that runs the program.

- GameDisplay.java
  This module creates the Game Displayed to the window of the game. It controls some of logic of the game, and calls other classes to generate the game board (so it controls the view).
  Interface
  Uses
  - Gameboard.java
  - PiecePanel.java

- Board.java
- FileIO.java
- GameData.java
- java.awt.*
- javax.swing.*
- javax.swing.border.*
- java.awt.event.ActionEvent
- java.awt.event.ActionListener
- java.awt.event.MouseAdapter
- java.awt.event.MouseEvent
- java.io.*
- java.io.PrintWriter
- java.util.Random

Return Type

- None

Access Programs

- public GameDisplay(String title) : A constructor for the GameDisplay class. Creates an GameDisplay object with a title, that is specified by String title. This method is necessary to allow other classes to create a game display, and run the program.
- public GameDisplay() :Creates an GameDisplay object, using the default settings and title. Calls the public method GameDisplay(String title) to create an instance of the Game Dispaly class.
- public void actionPerformed(ActionEvent e): A public method that overrides the more general actionPerformed. This method specifies the actions taken when the buttons in the button panel are used.

- GameBoard.java

  This module contains the data and specifications for displaying the 6 man's morris board to the window. It is used by Game Display to create the full game board.

  Interface

  Uses

  - java.awt.Color;
  - java.awt.Graphics;
  - java.awt.Graphics2D;
  - java.awt.Shape;
  - java.awt.geom.Ellipse2D;
  - java.awt.geom.Line2D;
  - java.awt.geom.Rectangle2D;
  - java.util.Random;
  - javax.swing.JPanel;

  Return Type

  - None

  Access Programs

  - public Shape[][] getShapeArray() : A public method to allow other class to access the values of the shapeArray 2D array. This allows other class to see the shape of each element on the board.
  - public void setVisibleTeams(int i, int j, int value) : A public method to allow other classes the ability to change the values of the visibleTeams array. This allows another class to change which player controls that space (has a piece on that space).

  - public int getVisibleTeams(int i, int j) : A public method that allows another class to get the value of the visibleTeams array at a specified i and j value. This allows another class to see which player has a piece on that space, if any.
  - public GameBoard() : A public constructor for the class. This allows another class to be able to create a GameBoard panel for the general display.

- PiecePanel.java
  This module contains the data and specifications for displaying the piece panel and messages to the window. It is used by Game Display to create the full game board.
  <u>Interface</u>
  <u>Uses</u>

  - java.awt.Color;
  - java.awt.Graphics;
  - java.awt.Graphics2D;
  - java.awt.Shape;
  - java.awt.geom.Ellipse2D;
  - javax.swing.JPanel;
  - javax.swing.JLabel;

  <u>Return Type</u>

  - None

  <u>Access Programs</u>

  - public Shape getRedCircle() : returns the value of RedCircle.
  - public Shape getBlueCircle() : returns the value of BlueCircle.

  - public void setLabel(String newText) : Allows another class to set the String message to be displayed to the screen.
  - public void setRedCount(String newText) : Allows another class to set the amount of remaining red disks to be placed.
  - public void setBlueCount(String newText) : Allows another class to set the amount of remaining blue disks to be placed.
  - public PiecePanel() : A constructor that allows another class to create an instance of the PiecePanel class.

- robut.java
  This module acts as an Artificial Intelligence to determine what move the computer should make. It is used by GameDisplay.java to give the computers next move based on a given board
  <u>Interface Uses</u>

  - none

  <u>Return Type</u>

  - Information on the game currently being played

  <u>Access Programs</u>

  - public static int[] place(int[][] visibleTeams, int colour, int added) : A public method that, based on the setup of the board and the number of pieces on it, determines and returns a location on the board where a new piece should be placed.
  - public static int[] move(int[][] visibleTeams, int colour) : A public method that, based on the setup of the board, determines and returns a set of locations on the board of what piece should be moved and where it should be moved to.
  - public static int[] mill(int[][] visibleTeams, int colour, int added) : A public method that, based on the setup of the board determines

and returns the location of a piece on the board that should be removed.

- GameData.java
  This module acts as the module in MVC framework, and stores and returns the information for the game currently in play. It is used by GameDisplay.java and Gameboard.java to return and update game data, and by FileIO.java return game data to be saved.
  Interface Uses
  - FileIO.java

  Return Type
  - Information on the game currently being played

  Access Programs
  - public GameData() : A constructor for GameData class. Allows another class to create Game Data for a new game.
  - public int[][] getVisibleTeams() : A getter method for the locations of the pieces on the game board.
  - public void setVisibleTeams(int x, int y, int value) : A setter method for an individual value of visibleTeams. Allows a piece to be added to the gameboard.
  - public void setVisibleTeams(int[][] newTeams) : A setter method for the whole of visibleTeams. Allows for an old game to be loaded to the game board.
  - public void incrementPlay1count() : Increments the number of pieces played by player 1 by 1.
  - public void decreasePlay1count() : Decrements the number of pieces played by player 1 by 1.

- public void setPlay1count(int count) : A setter method for the number of pieces played by player 1. Allows for an old games count of player 1s pieces to be loaded.
- public int getPlay1count() : A getter method for the number of pieces played by player 1.
- public int getPlay1removed() : A getter method for the number of player 1s pieces that were removed.
- public void incrementPlay2count() : Increments the number of pieces played by player 2 by 1.
- public void decreasePlay2count() : Decrements the number of pieces played by player 2 by 1.
- public void setPlay2count(int count) : A setter method for the number of pieces played by player 2. Allows for an old games count of player 2s pieces to be loaded.
- public int getPlay2count() : A getter method for the number of pieces played by player 2.
- public int getPlay2removed() : A getter method for the number of player 2s pieces that were removed.
- public void setRedTake(bolean value) : A setter method for whether or not red can take one of blues pieces.
- public Boolean getRedTake() : A getter method for whether or not red can take one of blues pieces.
- public void setBlueTake(bolean value) : A setter method for whether or not blue can take one of reds pieces.
- public Boolean getBlueTake() : A getter method for whether or not blue can take one of reds pieces.
- public void incrementTake() : A method that switches which player is active.

  - public void setFirst(boolean bool) : A setter method that sets which player moves first.
  - public boolean getFirst() : A getter method that returns which player moves first.
  - public void setState(int state) : A setter method that sets the previous state to the current state and sets the current state to the new state.
  - public int getCurrentState() : A getter method that returns the current state of the game.
  - public int getPreviousState() : A getter method that returns the previous state of the game.
  - public int getPreviousMoves() : A getter method that returns the number of moves previously made.
  - public void resetPreviousMoves() : A method that sets the number of previous moves to 0.
  - public void incrementPreviousMoves() : A method that increases the number of previous moves by 1.
  - public void loadGame() : A method that loads a game state from a txt file.

- FileIO.java
  This module saves the current game and loads the previous saved game.
  Interface
  Uses
  - java.io.BufferedReader
  - java.io.File
  - java.io.FileReader
  - java.io.PrintWriter

Return Type

- None

Access Programs

- public static void saveGame(GameData gameData): A public method which writes the current game data to a text file, allowing the user to save their progress.
- public static int[][] loadGame(): A public method which reads from an exsisting text file, allowing the user to return to a saved point in a game they already started.

- Board.java
  This module creates the game logic, and contains the error and logic checkers for the game.
  Interface
  Uses

  - GameDisplay.java
  - GameData.java
  - Boardreturn.java
  - javax.swing.SwingUtilities
  - java.util.Random;

  Return Type

  - None

  Access Programs

  - public boolean startboard(int[][][] visibleteams) : Sets up the array to keep track of piece positions on board, generates random boolean to determine which player goes first. Returns boolean (true - player1, false - player 2).

  - public void newpiece(int l, int p, boolean teamnew, int visibleteams[][], int play1count, int play2count): Adds new piece to array as long as there is no piece already in position and tracks how many pieces from each player have been added to board. Returns array of current board, and int value determining whether new piece position was legal (1 - legal).
  - public Boardreturn okaymovecheck(int team) : Determines whether any piece of the inputted team is able to move. If not, returns a 0, and player looses.
  - public static Boardreturn movepiece(int ol,int op,int nl,int np, int[][] visibleteams) : Holds values determining whether or not move/new piece placement is legal, and current array after new piece/move piece attempted.
  - public static int checkpiece(int oldl, int oldp, int newl, int newp, int[][] visibleteams): Checks whether proposed piece movement is legal based on current position, and position of other pieces on board.

- Boardreturn.java
  This module creates a return type to help determine if a boardstate is legal.
  Interface
  Uses

  - None

  Return Type

  - None

  Access Programs

  - public void setokayboard(int[][] boardchange): holds the current 'visibleteams' array after a move/new piece attempted

  - public void setmovestat(int status) : holds int value determing whether attempted move/new piece legal
  - public int[][] getokayboard() : returns current 'visibleteams' array when called
  - public int getmovestat() : returns whether previous attempted newpiece/movepiece was legal

3.2 MID

- GameMain.java
  Variables
  - None

  Access Programs
  - public static void main(String args[]): A public method which starts the game. It creates a game display object and directs it to the screen.

  Access Programs
  - None

- GameDisplay.java
  Variables
  - private static boolean isButtonNewGame : A boolean to show keep track if this is a new game.
  - private static boolean isButtonTake : A boolean to show if a button is selected.
  - private static boolean isButtonRecieve : A boolean to show if a button is received

- private PiecePanel piecePanel : A PiecePanel class to represent the pieces panel section of the display.
- private static GameBoard gamePanel : A GameBoard class to represent the game board section of the display.
- private int levels : An integer representing the amount of levels in the game. For 6 Man's Morris this is 2.
- private int places : An integer representing the amount of places in one level of the board. For 6 Man's Morris this is 8.
- private int[][] current : A 2D integer array representing the amount of pieces currently on a space of the board. Should only allow one piece on each space
- private int [] prevDisk : A 1D integer array representing the location of the piece that will be moved.
- private boolean aiOn : A boolean represent if the current game is being played against the AI.
- private int aiPlayer : An integer representing the current colour of the AI player.
- private int[] aiTarget : An integer array representing the target of the ai's actions. Could be where to place or move a piece on the board.

Access Programs

- public GameDisplay(String title): A constructor for the GameDisplay, that takes a String for the title of the game.
- public GameDisplay() : A generic constructor for the Game Display, calls GameDisplay(String title) to construct the game display.
- public void actionPerformed(ActionEvent e) : A public method representing the actions taken when a button is pressed on the board.

<u>Private Programs</u>

- private void state0() : A private method to be called when the game transitions to the first state (intial state).
- private void state1() :A private method to be called when the game transitions to the second state (piece placing).
- private void state2() :A private method to be called when the game transitions to the third state (piece moving).
- private void state3() :A private method to be called when the game transitions to the fourth state (milling).
- private void clearBoard() : A private method that is called that restarts the board, and returns to state0.
- private boolean checkMill(int i,int j,int colour): A private method that checks if a mill has been achieved, returns a boolean showing if there is a mill.

• GameBoard.java
<u>Variables</u>

- private Shape[][] shapeArray : A 2D Shape array containing the shapes at each of the nodes on the game board.
- private int[][] visibleTeams : A 2D integer array containing the values for the currrent controllers of a node on the gameboard (0-empty,1-red,2-blue).
- private int [][] sizingArray :A 2D integer array containing the values of the sizes of nodes on the game board. Keeps track of the scaling of each level.
- private double height : A private integer variable representing the height of the gameboard.
- private double width : A private integer variable representing the width of the game board.

Access Programs

  - public Shape[][] getShapeArray() : A public method allowing another class access to the values of the shapeArray variable array. Allows another class to see the shapes of each element on the game board.
  - public GameBoard() : A public constructor method that allows another class to create an instance of the GameBoard class.

Private Programs

  - protected void paintComponent(Graphics g) : An overrided method to describe how each element of game board will be coloured.

- PiecePanel.java
  Variables

  - private Shape redCircle : A private shape variable representing the shape of the red disk to be placed.
  - private Shape blueCircle:A private shape variable representing the shape of the blue disk to be placed.
  - private boolean blueTake: A private boolean variable representing if it is currently blue's turn.
  - private boolean redTake:A private boolean variable representing if it is currently red's turn.
  - private static boolean isButtonAdded:A private boolean representing if a button is to be added.
  - private static boolean redAddedLast: A private boolean representing if red was added last.
  - private static boolean blueAddedLast: A private boolean representing if blue was added last.

  - private JLabel label1: A Jlabel that contains the message to be displayed to the screen.
  - private JLabel label2: A Jlabel that contains the amount of red pieces remaining to be placed.
  - private JLabel label3: A Jlabel that contains the amount of blue pieces remaining to be placed.

  Access Programs

  - public Shape getRedCircle() : returns the value of RedCircle.
  - public Shape getBlueCircle() : returns the value of BlueCircle.
  - public void setLabel(String newText) : Allows another class to set the String message to be displayed to the screen.
  - public void setRedCount(String newText) : Allows another class to set the amount of remaining red disks to be placed.
  - public void setBlueCount(String newText) : Allows another class to set the amount of remaining blue disks to be placed.
  - public PiecePanel() : A constructor that allows another class to create an instance of the PiecePanel class.

  Private Programs

  - protected void paintComponent(Graphics g) : An overrided method to describe how each element of piece panel will be coloured.

- FileIO.java
  Variables

  - None

  Access Programs

  - public static void saveGame(GameData gameData): A public method which writes the current game data to a text file, allowing the user to save their progress.
  - public static int[][] loadGame(): A public method which reads from an exsisting text file, allowing the user to return to a saved point in a game they already started.

  Private Programs

  - None

- Board.java
  Variables

  - private int levels : A variable that holds number of levels on board
  - private int places : A variable that holds number of places in each level
  - private boolean first : A boolean that holds which player goes first

  Access Programs

  - public boolean startboard() : Sets up the array to keep track of piece positions on board, generates random boolean to determine which player goes first. Returns boolean (true - player1, false - player 2)
  - public void newpiece(int l, int p, boolean teamnew): Adds new piece to array as long as there is no piece already in position and tracks how many pieces from each player have been added to board. Returns array of current board, and int value determining whether new piece position was legal (1 - legal).
  - public Boardreturn okaymovecheck(int team) : Determines whether any piece of the inputted team is able to move. If not, returns a 0, and player looses.

- public static Boardreturn movepiece(int ol,int op,int nl,int np, int[][] visibleteams) : Holds values determining whether or not move/new piece placement is legal, and current array after new piece/move piece attempted.
- public static int checkpiece(int oldl, int oldp, int newl, int newp, int[][] visibleteams): Checks whether proposed piece movement is legal based on current position, and position of other pieces on board.

<u>Private Programs</u>

- private static Boardreturn checkmove(int oldl,int oldp) : Determines whether a move in the given position has any moves available to it. Checks which positions are adjacent, and whether or not there are already pieces there. Returns a 1 if moves available as well as an array of available moves, otherwise returns 0.

Tabular Expression of checkmove:

			Result:
If pos[x][y] = cur-pos[x][y+1]		Okay = 1	
Else	If pos[x][y] = curpos[x][y-1]		Okay = 1
	Else	If pos[x][y] = cur-pos[x+1][y] && (pos[x][y] % 2) != 0	Okay = 1
		Else	Okay = 0

Pre/Post Conditions for checkmove

Pre 0¡= x ¡= 7 && 0¡= y ¡= x

Post (Okay = 1) — (Okay = 0)

- BoardReturn.java

  Variables

  – private int updatedboard[][] : A variable that holds the value of the updated board.

  – private int movestat : An integer to represent if the move is legal.

  Access Programs

  – public void setupdatedboard(int[][] boardchange): holds the current 'visibleteams' array after a move/new piece attempted

  – public void setmovestat(int status) : holds int value determing whether attempted move/new piece legal

  – public int[][] getupdatedboard() : returns current 'visibleteams' array when called

  - public int getmovestat() : returns whether previous attempted newpiece/movepiece was legal

  Private Programs

  - None

- robut.java

  Variables

  - None

  Access Programs

  - public static int[] place(int[][] visibleTeams, int colour, int added) : A public method taking 3 inputs: visibleTeams, an integer array of the setup of the board; colour, an integer of what which teams piece is being placed; added, an integer of the number of pieces added to the board. Determines and returns a location on the board where a new piece should be placed.

Tabular Expression of place() :

				Result:
If added = 0				target = random()
Else	If checkNearMill[0] != null			target = checkNearMill
	Else	If checkMill[0] != null		target = checkMill
		Else	If checkAdjacent != null	target = checkAdjacent
			Else	target = random()

Pre/Post Conditions for checkmove

Pre {added ¿= 0 }

Post (target = random())—(target = checkNearMill)—(target = checkMill)—(target = checkAdjacent)

- public static int[] move(int[][] visibleTeams, int colour) : A public method taking 2 inputs: visibleTeams, an integer array of the setup of the board; colour, an integer of what which teams piece is being placed. Determines and returns a set of locations on the board of what piece should be moved and where it should be moved to.
- public static int[] mill(int[][] visibleTeams, int colour, int added) : A public method taking 2 inputs: visibleTeams, an integer array of the setup of the board; colour, an integer of what which teams piece is being placed. Determines and returns the location of a

piece on the board that should be removed.

Private Programs

- private static int[] random(int[][] visibleTeams, int search) : A private method that searches for a random piece of a certain colour on the board and returns its location.
- private static in[][] nearMill(int[][] visibleTeams, int colour, int action) : A private method that searches for any near mills of a certain colour and either returns and array of integer arrays of either the empty spots of near mills to be completed or pieces in near mills to be removed.
- private static int nearMillRandom(int empty, int pieceA, int pieceB, int action) : A private method that, based on the value of action, either returns the location of the empty spot of a near mill or one of the two pieces in a near mill.
- private static int[][] checkAdjacent(int[][] visibleTeams, int adj, int i, int j) : A private method that, given a location and a type to search for (a piece of a certain colour or an empty spot), will return an array of integer arrays of the locations of the adjacent pieces of that type.
- private static int[][] checkAdjacent(int[][] visibleTeams, int adj, int i, int j) : A private method that, given a location and a type to search for (a piece of a certain colour or an empty spot), will return an array of integer arrays of the locations of the adjacent pieces of that type.
- private static int[][] findMill(int[][] visibleTeams, int notColour) : A private method that searches for any mills that a player has and returns one piece from each mill to be removed.
- private static int findMillRandom(int pieceA, int pieceB, int pieceC) : A private method that, given a mill, randomly returns one of the

three pieces to be removed.

  - private static int[] nearbyPiece(int[][] visibleTeams, int[][] nearMills, int search) : A private method that, given an array of locations of near mills, will search for and return the location of a nearby piece of a certain colour to block or complete the near mill, if one exists.

- GameData.java
  Variables

  - private int[][] visibleTeams : An integer array of the current state of each piece.
  - private boolean redTake : A boolean to mark whether or not red is the active player. If true, red is active. If not, red is not active.
  - private boolean blueTake : A boolean to mark whether or not blue is the active player. If true, blue is active. If not, blue is not active.
  - private boolean first : A boolean that represents which player goes first.
  - private int currentState : An integer to mark which state the game is currently in.
  - private int previousState : An integer to mark which state the game was in before it became its current state.
  - private int play1count : The number of active pieces for player 1.
  - private int play2count : The number of active pieces for player 2.
  - private int play1removed : The number of pieces of player 1s that have been removed.
  - private int play2removed : The number of pieces of player 2s that have been removed.

- private int previousMoves : The number of moves that have been made in the game.
- private final int allowable : The maximum number of pieces each player is allowed to have.

Access Programs

- public GameData() : A constructor for GameData class. Allows another class to create Game Data for a new game.
- public int[][] getVisibleTeams() : A getter method for the locations of the pieces on the game board.
- public void setVisibleTeams(int x, int y, int value) : A setter method for an individual value of visibleTeams. Allows a piece to be added to the gameboard.
- public void setVisibleTeams(int[][] newTeams) : A setter method for the whole of visibleTeams. Allows for an old game to be loaded to the game board.
- public void incrementPlay1count() : Increments the number of pieces played by player 1 by 1.
- public void decreasePlay1count() : Decrements the number of pieces played by player 1 by 1.
- public void setPlay1count(int count) : A setter method for the number of pieces played by player 1. Allows for an old games count of player 1s pieces to be loaded.
- public int getPlay1count() : A getter method for the number of pieces played by player 1.
- public int getPlay1removed() : A getter method for the number of player 1s pieces that were removed.
- public void incrementPlay2count() : Increments the number of pieces played by player 2 by 1.

- public void decreasePlay2count() : Decrements the number of pieces played by player 2 by 1.
- public void setPlay2count(int count) : A setter method for the number of pieces played by player 2. Allows for an old games count of player 2s pieces to be loaded.
- public int getPlay2count() : A getter method for the number of pieces played by player 2.
- public int getPlay2removed() : A getter method for the number of player 2s pieces that were removed.
- public void setRedTake(bolean value) : A setter method for whether or not red can take one of blues pieces.
- public Boolean getRedTake() : A getter method for whether or not red can take one of blues pieces.
- public void setBlueTake(bolean value) : A setter method for whether or not blue can take one of reds pieces.
- public Boolean getBlueTake() : A getter method for whether or not blue can take one of reds pieces.
- public void incrementTake() : A method that switches which player is active.
- public void setFirst(boolean bool) : A setter method that sets which player moves first.
- public boolean getFirst() : A getter method that returns which player moves first.
- public void setState(int state) : A setter method that sets the previous state to the current state and sets the current state to the new state.
- public int getCurrentState() : A getter method that returns the current state of the game.

- public int getPreviousState() : A getter method that returns the previous state of the game.
- public int getPreviousMoves() : A getter method that returns the number of moves previously made.
- public void resetPreviousMoves() : A method that sets the number of previous moves to 0.
- public void incrementPreviousMoves() : A method that increases the number of previous moves by 1.
- public void loadGame() : A method that loads a game state from a txt file.

Private Programs

- None

4 Traceability

Requirement	Module	Result
Current Game Can be Saved	FileIO	Current game data is saved to a .txt file
Previous Game Can be Loaded	FileIO	.txt file containing game state previously saved is read, and sets current game data from .txt file.
Random Player selected to go First	Board	random boolean generated to represent either player 1 or player 2
Setting up Board Array	Board	int[][] generated to hold values at each position on array. Initially, the array holds all zeros, representing no disks on board.
Checking if New Piece Position is Legal	Board	If the position is legal (no other pieces already in that position), returns a 1 and the new array, otherwise returns a 0 and the old array.

Requirement	Module	Result
Checking if Moved Piece Position is Legal	Board	If the position is legal (no other pieces in that position, it is an adjacent position), returns a 1 and the new array, otherwise returns a 0 and the old array.
Checking for Win Condition	Board	If there are no moves available to the player, returns a 0, and player looses the game.
Displaying an array as a Six Men's Morris Board	GameBoard	Game is Displayed as a panel on a jFrame, allows for user interaction
Allowing the user to place a piece	GameBoard	User is able to select a location and place a piece
The pieces used in the game are red and blue	GameBoard	The gameboard draws the pieces and restricts their colors to red and blue
Starting with an empty board displayed	GameBoard	All of the spaces are initial empty (black) on the gameboard.

Requirement	Module	Result
New Game Button	GameDisplay	Allows user to select option to restart the board, will call GameControl.
Check Button	GameDisplay	Allows user to check if current board is legal, will call GameControl.
User-Game Interaction	GameDisplay	Allows user to interact with game pieces, make modifications to current board, will call GameControl.
Game is able to recognize a Winner	Game Display	Game has criteria to meet (player only has 2 pieces left) to determine if there is a winner
Players are only able to make legal moves	BoardReturn	Game is able to determine which moves are legal and only allows player to make correct moves
Players make moves in turn	Game Display	Game alternates player turns, and displays the current player to the screen

Requirement	Module	Result
Players are able to draw	Game Display	Game will result in a draw after 16 consecutive moves without a mill.
Players are able to select to play against a computer (AI) player.	Game Display	There is a button to start a game against an AI player, which will start a game where one player (randomly determined) is a computer.
The AI is able to place a piece	robut: place()	The AI is able to determine where on the board it should place a piece
The AI is able to move a piece	robut : move()	The AI is able to determine where it is able to move a piece
The AI is able to mill an oppenents piece()	robut : mill()	The AI is able to determine the best piece it should remove from the player, and it successfully removes this piece.

5 Uses Relation

In the above relation, a class points to another that it uses.
GameMain only uses GameDisplay, as it only needs to create a GameDisplay

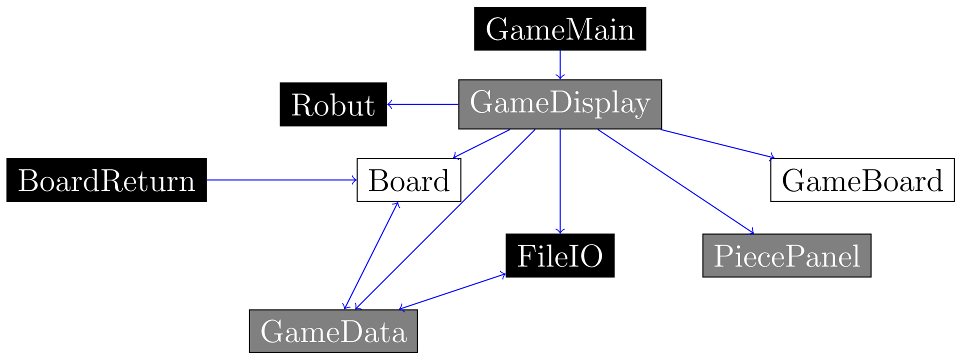

Figure 3: Uses Relation

object to run the program.
GameDisplay uses PiecePanel and Gameboard to create the panels on the window that are displayed. It uses FileIO to load and save game information to the file. It uses Board to determine game logic and apply it to the objects in the window. It uses GameData to determine the colors and what messages to display to the screen. It uses Robut to determine the actions the AI should take when playing the game.
Board uses two classes BoardReturn and GameData. It uses BoardReturn to help return information to GameDisplay. It uses GameData to determine what it should tell GameDisplay to display (The controller telling the view what to do). GameData uses GameDisplay, Board and FileIO. It uses these to determine what it should change in its internal variables. It represents the model of the architecture, so must interact with both the controller and the view.

6 Testing

6.1 Informal Black Box Testing

What was Tested	What it did	Comments
GameControl: startboard	Printed out 'visibleteams' array	startboard is correctly setting up the 'visibleteams' array
GameControl: startboard	Printed out "first" boolean	"first" is correctly assigned random boolean values (on 3 separate tests was assigned true, false, and false)
GameControl: newpiece	Entered various input values (ie: 1,1,true)	Values were correctly entered into 'visibleteams' (ie: visibleteams[1][1] = 1)
GameControl: newpiece	Was unable to place new peices in spots already occupied (ie: visibleteams[1][1] = 1, then a newpeice was entered there)	An int value of 0 was returned (meaning the move was not legal) and the array was unchanged.
GameControl: movepiece	Ran function while no illegal moves attempted	Correctly returned no errors, updated array correctly
GameControl: movepiece	Attempted to move pieces illegally on top of each other (ie: piece at [0][0] moved ontop of another piece at [0][1])	Returned 0, no changes made to 'visibleteams' array (piece remained at [0][0], no other pieces were moved)

What was Tested	What it did	Comments
GameControl: movepiece	Attempted to move pieces illegally to positions not adjacent to the current position	Returned 0, no changes made to ’visibleteams’ array (piece remained at [0][0])
GameBoard: GameBoard()	Display the board, allow for color change of disks	Constructor Method for the gameboard panel. Displays the board as a set of pieces.
GameBoard: GameBoard() Change Piece Color	Pieces Change color	The display was successfully able to read in the current color array.
GameDisplay: GameDisplay()	Displayed panel in correct position in frame with buttons in correct positions in panel	GameDisplay is correctly setting up the panel
GameDisplay: GameDisplay()	Clicked on ”New Game?” Button, printed ”New Game!” in command line	GameDisplay is correctly running through correct output for the given input
GameDisplay: GameDisplay()	Clicked on ”Check!” Button, printed ”Is is correct?” in command line	GameDisplay is correctly running through corect output for the given input
PiecePanel: PiecePanel()	Displayed panel in correct position in frame with circle tokens in correct position in panel	PiecePanel is correctly setting up the panel

What was Tested	What it did	Comments
PiecePanel: PiecePanel()	Clicked on Red Circle, Displayed ”add red?”	PiecePanel is giving the correct output for the given input and is ready to add a red piece to the board
PiecePanel: PiecePanel()	Clicked on Blue Circle, Displayed ”add blue?”	PiecePanel is giving the corect output for the given input and is ready to add a blue piece to the board
PiecePanel: PiecePanel() , Clicked on Red Circle, then white circle	White Cirtcle turned red	PiecePanel is giving the corect output for the given input and should add a red piece to the board
PiecePanel: PiecePanel() , Clicked on Red Circle, then white circle that has been coloured red	No output occurred	PiecePanel is giving the corect output for the given input and any additional red peices were not added not on their turn
PiecePanel: PiecePanel() , Clicked on Blue Circle, then white circle that has been coloured red	The red coloured white circle turned blue	PiecePanel is giving the corect output for the given input and a blue piece should be added
PiecePanel: PiecePanel() , Clicked on Blue Circle, then white circle	White Circle turned Blue	PiecePanel is giving the corect output for the given input and should add a blue piece to the board
PiecePanel: PiecePanel() , Clicked on Blue Circle, then white circle that has been coloured blue	No output occurred	PiecePanel is giving the corect output for the given input and any additional red peices were not added not on their turn

What was Tested	What it did	Comments
PiecePanel: PiecePanel() , Clicked on Red Circle, then white circle that has been coloured blue	The blue coloured white circle turned red	PiecePanel is giving the corect output for the given input and a red piece should be added
GameDisplay	Attempted to add pieces to board	Pieces were added correctly. Game state (adding pieces) was displayed until each player had 6 pieces on board, turn system working correctly, and current turn was displayed. It was not possible to add pieces on top of each other.
GameDisplay	Attempted to move pieces across board	Pieces were only able to move to legal positions (adjacent), no flying was possible. Which players turn it was and when a piece was selected were displayed.
GameDisplay	Attempted to mill	When pieces were milled, state was displayed. Successfully deleted other players piece when mill formed. Game continued to function thereafter.

What was Tested	What it did	Comments
GameDisplay	Attempted to win game	When player is down to 2 pieces, the other player's winning status is displayed.
GameDisplay	Attempted to save and load game	Game saved and reloaded correctly, all pieces where they were left.

NEW

What was Tested	What it did	Comments
FileIO: saveGame and loadGame	Played partway through a game, attempted to save, exit program and continue playing	Game continued from where it was saved, all pieces remained in the same positions.
Board: startboard	Printed out ’visibleteams’ array	startboard is correctly setting up the ’visibleteams’ array
Board: startboard	Printed out ”first” boolean	”first” is correctly assigned random boolean values (on 3 separate tests was assigned true, false, and false)
Board: newpiece	Entered various input values (ie: 1,1,true)	Values were correctly entered into ’visibleteams’ (ie: visibleteams[1][1] = 1)

What was Tested	What it did	Comments
Board: newpiece	Was unable to place new peices in spots already occupied (ie: visibleteams[1][1] = 1, then a newpeice was entered there)	An int value of 0 was returned (meaning the move was not legal) and the array was unchanged.
Board: movepiece	Ran function while no illegal moves attempted	Correctly returned no errors, updated array correctly
Board: movepiece	Attempted to move pieces to each position on each level consecutively	Error occured. Piece was unable to move from the 7th to 0th place on level. Correction was made to identation of movepiece function. A second test of the same nature was performed, and pieces moved legally and correctly.
Board: movepiece	Attempted to move pieces illegally on top of each other (ie: piece at [0][0] moved ontop of another piece at [0][1])	Returned 0, no changes made to 'visibleteams' array (piece remained at [0][0], no other pieces were moved)

What was Tested	What it did	Comments
Board: checkmove	Set up board, and ran function for each piece placed	An error occured (ie: the position array [[2, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0, 2]], available moves for piece in position [0][0] : [[0, 1, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 0, 0]]) Correction to checkmove made. Test repeated, successful.
Board: okaymovecheck	Set up board, when team had an available move	Returned a 1, test successful.
Board: okaymovecheck	Set up board, when team had no available move	Returned a 0, test successful.
Board: movepiece, checkpiece	Attempted to move pieces illegally to positions not adjacent to the current position	Returned 0, no changes made to 'visibleteams' array (piece remained at [0][0])

What was Tested	What it did	Comments
GameDisplay: Starting a game against an AI	A game began and one of the players was controlled by the game	This was the expected result.
robut: place(), with input visibleTeams = 0,0,2,0,0,0,1,1, 0,0,0,0,0,0,0,0 , 1 , 3	Output location was 0,0	Place() is functioning correctly
robut: place(), with input visibleTeams = 0,0,2,0,0,0,1,1, 0,0,0,0,0,0,0,0 , 2 , 3	Output location was 0,0	Place() is functioning correctly
robut: place(), with input visibleTeams = 0,0,2,0,0,0,0,0, 0,0,0,0,0,2,1,2 , 1 , 2	The function got stuck in an infinite loop	Changed randomAdjacent() so it does not randomly select a piece but instead it uses a for loop to check for a piece with and adjacent empty space
robut: place(), with input visibleTeams = 0,0,2,0,0,0,0,0, 0,0,0,0,0,2,1,2 , 1 , 2	Array out of bound exception	Changed randomAdjacent() so it either returns an array of the location or it returns null

What was Tested	What it did	Comments
robut: place(), with input visibleTeams = 0,0,2,0,0,0,0,0, 0,0,0,0,0,2,1,2 , 1 , 2	Output location was 0,4	Place() is functioning correctly
robut: move(), with input visibleTeams = 0,1,2,1,0,2,2,2, 0,1,1,2,0,0,0,0 , 2	Output piece and location were 0,7 and 0,0	Move is not functioning correctly, added catch so pieces in near mills cannot move to try complete themselves
robut: move(), with input visibleTeams = 0,1,2,1,0,2,2,2, 0,1,1,2,0,0,0,0 , 2	Output piece and location were 1,3 and 1,4	Move is functioning correctly
robut: move(), with input visibleTeams = 0,1,2,1,0,2,2,2, 0,1,1,2,0,0,0,0 , 1	Output piece and location were 0,1 and 0,0	Move is functioning correctly
robut: mill(), with input visibleTeams = 0,1,2,1,0,2,2,2, 0,1,1,2,0,0,0,0 , 1	Output was 0,6	Mill is functioning correctly

What was Tested	What it did	Comments
robut: mill(), with input visibleTeams = 0,1,2,1,0,2,2,2, 2,1,1,2,0,0,0,0 , 1	Output was 0,0	Mill is functioning correctly
Board was 2,1,2,1,2,0,2,1, 0,1,2,1,2,0,1,0, piece at 0,7 was clicked and was attempted to be moved to 1,7	Piece could not be moved to 1,7. Random clicking led to the piece finally being moved to 1,0	Game is not functioning correctly
Playing game with AI, Board was 1,0,2,2,2,0,1,2 1,1,2,0,1,1,2,1, player (2) has mill and tried to remove a 1 piece	Piece could not be removed, blue got an extra turn	Game is not functioning correctly
Playing game with AI, Board was 1,0,2,2,0,2,1,2 1,1,2,0,1,1,2,1, player (2) moved piece at 0,5 to 0,4 to create mill	Game did not recognize mill, instead game was ended in draw	Game is not functioning correctly
Playing game with AI, Board was 0,0,1,2,1,1,1,0 2,1,1,2,2,0,0,2, computer (1) has mill	AI did not remove piece, instead got an extra turn to move	Game is not functioning correctly

7 Computer Player AI

The computer player was designed with the three states of gameplay in mind. Just as the player was implemented through parsing inputs in differing methods depending on the state of the game, the computer was implemented as an abstract class that returns inputs of where to move next depending on which state the game is in. For each state there is a corresponding function to determine what move the computer player should make: place() corresponds with state 1; move() corresponds with state 2; mill() corresponds with state 3.

The computer player was designed with the three states of gameplay in mind. Just as the player was implemented through parsing inputs in differing methods depending on the state of the game, the computer was implemented as an abstract class that returns inputs of where to move next depending on which state the game is in. For each state there is a corresponding function to determine what move the computer player should make: place() corresponds with state 1; move() corresponds with state 2; mill() corresponds with state 3.

The function move() works in a similar fashion. It begins by checking if the computer has a near mill. If it has that it continues to check for whether there is one of the computers pieces adjacent to the empty spot of the near mill. If that is true, then the piece is moved to complete the mill. However, if any one of those are false move() checks if the other player has a near mill. If true, it continues to check for whether one of the computers pieces is adjacent to the empty spot of the opponents near mill. If that is also true, the piece is moved to block the other player from creating a mill. If all else fails, a random piece of the computers is moved to a random open adjacent location.

Finally, mill()s implementation follows that of move() and place(). It begins by checking if the opposing player has a mill and, if it does, randomly removes one of the three pieces that make up the mill. If the opponent does not have a mill, the function checks if they have a near mill instead and removes one of the pieces that makes it up, if it exists. Otherwise, mill() randomly removes one of the opponents pieces.

8 Discussion

The design follows the MVC format quite well. The model is implemented in GameData; the View is implemented in GameBoard, PiecePanel, and GameDisplay; and the Controller is implemented in Main and Board. Each class is much better balanced in terms of the tasks it should be doing and how it is interacting with the other classes, as compared to assignment 2. The code is properly modularized to follow MVC format and there is a clear hierarchy. The group would give this project an 8/10 due to the improvements made.

8.1 Anticipated Changes

- Board: In the future, should any additional rules be implemented in the game, it would not be difficult to integrate them with the already exsisting ones. Additionally, the Board could be easily used with different dimensions (ie: 8 Men's Morris).

- Boardreturn: Class designed to hold variables needed to determine whether a move/new piece is legal. Returns a integer indicating the legality, and a int[][], indicating the current board array. Due to the design of a separate type to bundle this data, if any other data feedback is required in the future, it will be simple to add it in.

- FileIO: In the future, the program could store additional data from the user (high score, trends, etc), by writing it to the .txt file.