Six Men's Morris
SFWR ENG 2AA4/ COMP SCI 2ME3
Tutorial 1 Group 5

Anthony Chang
Prajvin Jalan
Nicolas Tristani

Table of Contents

Module Decomposition.....2

Classes and Descriptions (public and private entities).....2

Computer AI Algorithm.....7

Uses Relationship.....8

Traceback to Requirements.....11

Internal Review/Evaluation.....12

Program Test Report.....13

All greyed section are from the last 2 assignments

Module Decomposition

Our modules were designed using a top-down approach and were first classified as either part of the user interface or the game logic. The classes MainStartUp, PointLabel, and SixGameView all help display the game and information to the user, while the classes Player, Point, SixGame, and SixMensMorris controls the logic of the game as well as the responses to any interactions.

As a result of using MVC architecture, our program successfully embodies the *separation of concerns* principle, ensuring that the logic of the game is unaffected by interface changes and vice versa. Since our modules are also built based off object-oriented design concepts, components within each module are protected from external entities so our program incorporates *information hiding* as well. Overall the way the modules are broken down allows for easy object manipulation and the game's logic is very efficient, which is why it is decomposed in such a way.

An additional class was created in order to create a save/load system. Since this is an entirely separate module, it ensures information hiding and allows easy alterations without affecting the main program.

Classes and Descriptions (public and private entities)

MainStartUp.java: The MainStartUp class displays the game to the user using JFrame. The Constructor, when called, creates the look of the window and adds two buttons to the display.

Public entities

- actionPerformed (ActionEvent e) - Handles what action to perform when a button is clicked
- setOldFrame(JFrame oldFrame) - Mutator for the frame that holds this window

Private entities

- JButton newGameButton and presetGameButton - These are the buttons that the user interacts with to start a new game or a preset game
- JFrame oldFrame - This is the frame that holds the first window the player sees (window containing the new and preset game buttons)

Player.java: The player class represents the players of the game. Two player classes are implemented for this game (player 1 and player 2) which contains information on which pieces they have placed on the board, and which pieces are unplaced.

Public entities

- getValue() - Returns value of player (1 for Player 1 or 2 for Player 2)
- getOtherPlayer() - Returns the value of opposing player
- getUnplaced() - Returns the player's unplaced pieces
- getPlaced() - Returns the player's placed pieces
- placePiece() - Subtracts 1 from unplaced pieces and adds 1 to placed pieces
- removeSetupPiece() - Adds 1 to unplaced pieces and subtracts 1 from placed pieces
- removeGamePiece() - Removes a piece on the board during removal phase of game
- getAI() - Returns if the player is an AI

Private entities

- int value - Holds a value that determines which player has the current turn (player 1 or player 2)
- int unplaced - Holds the amount of pieces that have yet to be placed
- int placed - Holds the amount of pieces that have been placed.
- Boolean ai - Turns AI control on/off

Point.java: The Point class represents a possible placement location for the player's pieces. Each Point object holds information on their location as well as value (if a piece is on the point and if so belonging to which player). There are 2 constructors for this class; the first sets up a generic point at a given location while the second is used if a specific value needs to be set.

Public entities

- getValue() - Returns the value of the point: empty, blue, or red
- getSquare() - Returns the square the point is located on (inner or outer)
- getLocation() - Returns the location of the point on the square
- setValue(int value) - Changes the value of this point (empty point (0), blue (1), red (2))
- setEnabled(boolean enabled) - Sets a point's enabled value
- getEnabled() - Returns if this point is enabled
- squareAdjacency(Point otherPoint) - Returns if points are adjacent to one another
- sameSquare(Point otherPoint) - Returns if the points are on the same square

Private entities

- int value - This holds a value representing what piece is placed. 0 for empty, 1 for player one, 2 for player two
- int square - Determines whether the point is in the inner or outer square
- int location - This holds the specific location of the point on the square
- boolean enabled - Determines whether a point is enabled or disabled

PointLabel.java: The PointLabel class helps graphically display the points and pieces to the players. The constructor associates a specific point object with a PointLabel to control the colour.

Public entities

- getPoint() - Returns the current label's point
- paintComponent(Graphics g) - Draws the appropriate point on the board
- setSelected(boolean selected) - Change the label's selection
- getSelected() - Returns if the label is selected

Private entities

- Point point - holds the point for the current label
- boolean selected - Determines if a point is selected on the board

SixGame.java: The SixGame class is responsible for instantiating the six men's morris game. The constructor creates either a new game or a game from a preset board and initializes the players and pieces.

Public entities

- getView() - Returns the game's view
- getCurrentPlayer() - Returns the current player on the board
- getPlayer(int value) - Returns the player associated to the value parameter
- getPoint(int square, int location) - Returns the point specified in the parameter
- getPlacedPieces(int player) - Returns the placed pieces of the specified player
- getUnplacedPieces(int player) - Returns the unplaced pieces of the specified player
- setPhase(Phase phase) - Sets current phase of the game
- getPhase() - Returns the current phase of the game
- getMills() - Returns if milling is occurring
- endTurn() - Ends the current players turn
- positionClicked(Point point) - Receives point clicked instructions and performs various actions based on current phase of the game
- millPieces(Point point) - Checks if there are 3 consecutive pieces at a point
- moveMade(Point selected, Point newLocation) - Makes a move and returns if the move was legal and done
- removePiece(Point selected) - Attempts to remove a player's piece and returns condition of the removal
    - Tabular Expression

Condition	**Result**
selected.getValue() == 0	false
selected.getValue() == this.getCurrentPlayer().getValue(	false

)	
selected.getValue() != 0	true*
selected.getValue() != this.getCurrentPlayer().getValue( )	true*

*selected's value changes to 0, point at selected is unfilled, piece removed from current player

- Pre/Post Condition
  {selected.getValue() == 0 V selected.getValue() == this.getCurrentPlayer().getValue()}
  removePiece(Point selected)
  {false}

- placePiece(Point point) - Places a piece on the board for the current player
- getCurrentPoint() - Returns the current point selected
- setCurrentPoint(Point point) - Sets the current point selected
  - Tabular Expression

Condition	**Result**
point == null	deselectPoints()
point != null	selectPoint()

  - Pre/Post Condition
    {point == null}
    setCurrentPoint(Point point)
    {deselectPoints()}
- setCurrentPlayer(Player player) - Sets the current player
- checkEnd() - Checks whether or not the game has been lost
- computerMove() - AI algorithm; controls placement, movement, and removal

Private entities

- SixGameView gameView - This is the game display
- Point[ ] [ ] points - This is a 2D array that holds all the points on the board
- Point currentPoints - This holds the currently selected points
- Player[ ] players - Holds a value for each player. 0 for current player, 1 for player one, 2 for player two
- enum Phase - Holds different phases of the game (placement, movement, removal)
- Phase gamePhase - Holds the current phase of the game
- boolean mills - Holds if milling is occuring
- setPoints(Point[][] points) - method called in constructor to set up points on the board

- initializePlayers(int pieces) - method called in constructor to initialize players with appropriate number of pieces

SixGameView.java: The SixGameView class controls the main interactions within the game such as mouse clicks.

Public entities

- fillPoint (int square, int location, int player) - Sets a point's colour as specified by parameters
- paintComponent (graphics g) - Draws the appropriate graphics on the view
- mouseClicked (MouseEvent e) - Handles what to do when the mouse is clicked
- actionPerformed (ActionEvent e) - Handles what to do when a button is clicked
- findErrors (boolean errorsExist) - Resets or creates a new frame when an error is found
- setOldFrame (JFrame oldFrame) - Mutator for the frame that holds this window
- setAllPointsEnabled() - Sets every point on the board to enabled
- setPointsDisabled(int player) - Sets appropriate pointLabels to disabled
- deselectPoints() - deselect points on the board that are selected
- selectPoint(int square, int location) - Selects a point based on its square and location
- endGame(int result) - Updates view to reflect ending of game
- mousePressed(MouseEvent e) - Override; currently unused
- mouseReleased(MouseEvent e) - Override; currently unused
- mouseEntered(MouseEvent e) - Override; currently unused
- mouseExited(MouseEvent e) - Override; currently unused

Private entities

- SixGame game - The viewed model
- Rectangle outerSquare, innerSquare - The outer and inner squares to be drawn
- PointLabel[ ] [ ] pointLabels - Labels that represent the points
- JButton blueButton, redButon, doneButton - Buttons for the setup board, respective to their color
- JLabel playerOneDisks, playerTwoDisks - Labels for each player's disk
- JTextArea setupRules, gameProgress, turnPlayer - This is the text area that holds the setup rules, Progress of the current game, and player's turn
- int currentColor - Current color disk being set on the setup board
- boolean setupBoard - Determines if the current board is a setup board or not
- JFrame oldFrame - Frame that contains this view

- ArrayList<String> errorList - Holds all errors that may occur during board setup
- setPointLabels() - creates point labels and adds them to the view
- setPointLabels(PointLabel[][] pointLabels) - overloaded method, creates preset point labels and adds them to the view
- setBoardPointLabels() - Sets the PointLabels for the entire board
- getPointLabel(int square, int location) - Returns the point label at specified square and location

SixMensMorris.java: The SixMensMorris class contains the main method used to run the game.

Public entities

- main (String[] args) - Main method; Starts the program

Private entities

- none

FileIO.java: Class reads and writes to a file; used for saving and loading a game

Public entities

- getFile(String type) - opens the appropriate dialog box; return selected file
- save() - export current game into a ".sav" file
- load() - imports a game from a ".sav" file
-

Private entities

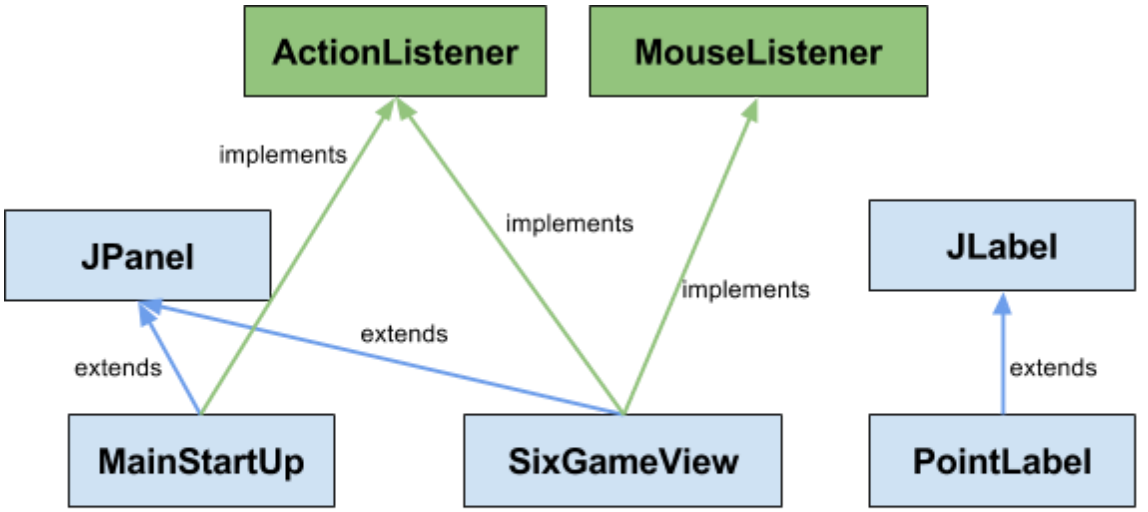
- JPanel fileContents - load file chooser box
- JFileChooser fileChooser - allows player to choose save/load file
- FileNameExtensionFilter filter - filter for files
- File myFile - File to save or load from
- SixGame gameModel - attached game model
- JFrame frame - When loading from a file

Computer AI Algorithm

The algorithm used in the program to determine which location to place a disk is random selection. The computer player will choose a location on the board and attempt to place/move his disk to that location. If the location is occupied or the move is illegal, the computer will randomly select a new location and try again until a valid position has been found. The AI can also remove a disk belonging to the player (randomly) if it gets a mill, and winning the game is possible if the player is reduced to two disks. Although this algorithm is not very complex, it does occasionally challenge players as the computer's movement is unpredictable.

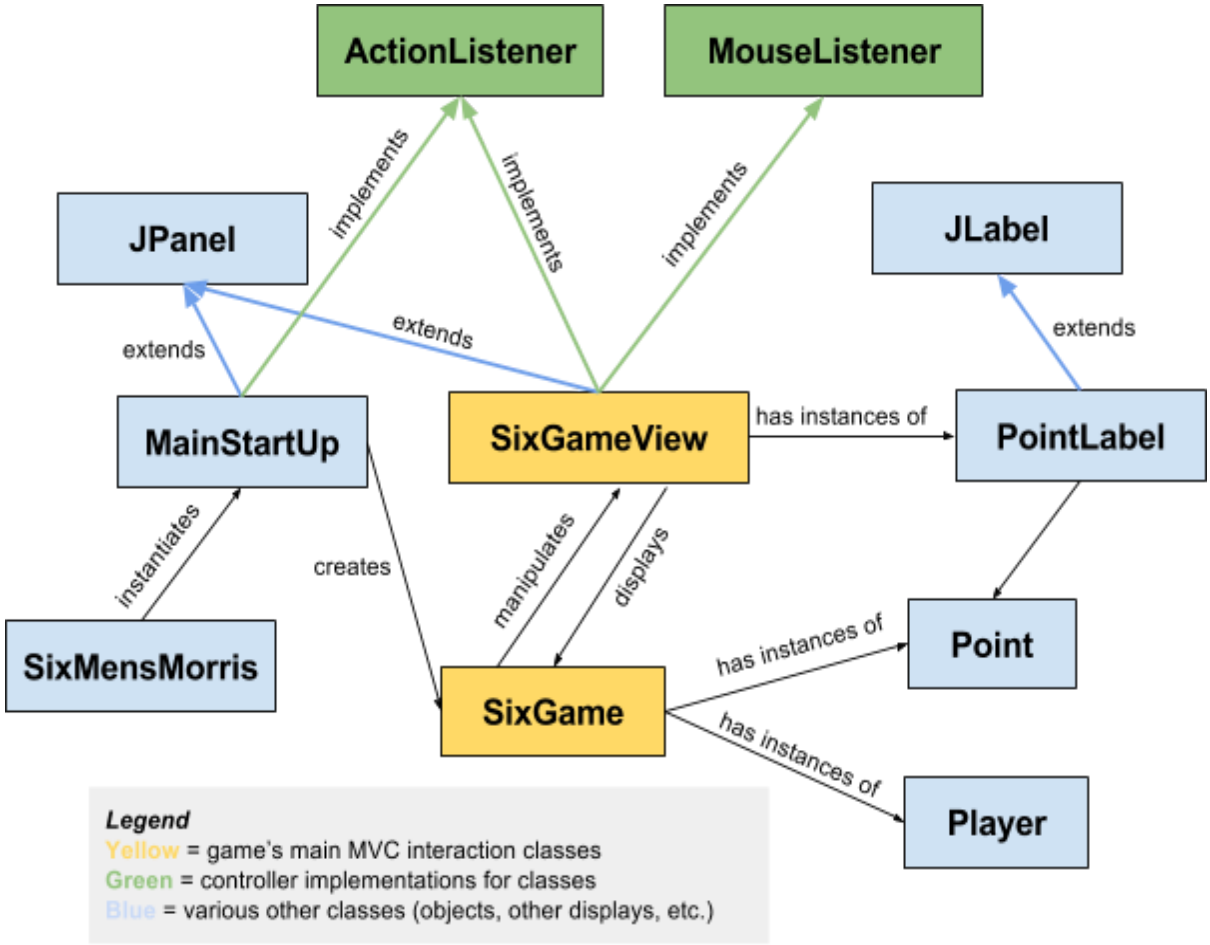
Uses Relationship

PREVIOUS Version 1



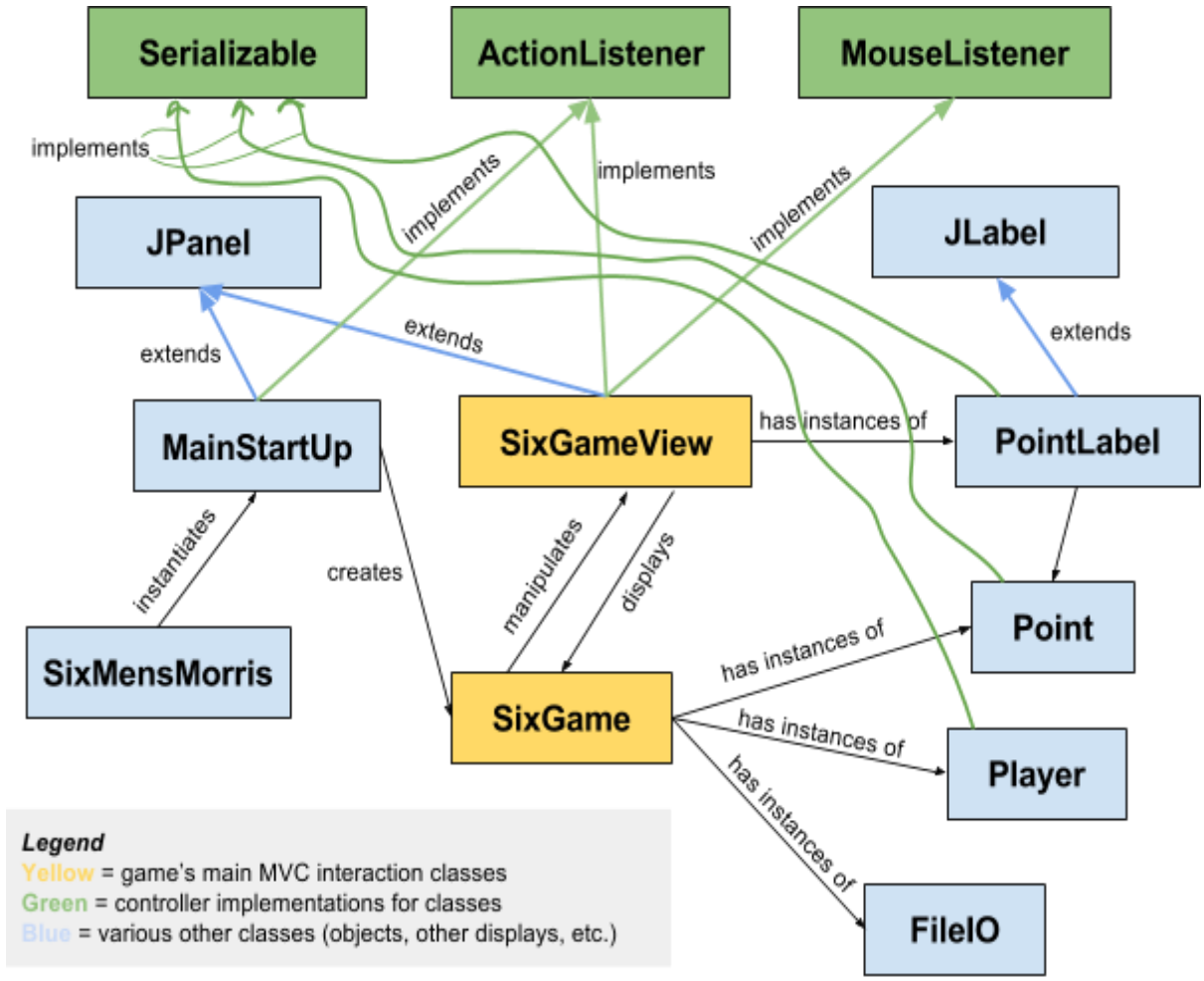
- ➔ *MainStartUp uses the JPanel and ActionListener modules*
- ➔ *SixGameView uses the JPanel, ActionListener and MouseListener modules*
- ➔ *PointLabel uses the JLabel module*

Previous Version 2



- ➔ *MainStartUp uses the JPanel and ActionListener modules*
- ➔ *SixGameView uses the JPanel, ActionListener and MouseListener modules*
- ➔ *PointLabel uses the JLabel module*
- ➔ *SixMensMorris class creates an instance of MainStartUp, to create the window that starts the game*
- ➔ *MainStartUp creates the SixGame module, and SixGame creates the SixGameView module*
- ➔ *SixGame manipulates SixGameView, which displays SixGame's information (SixGame is model, SixGameView is view/controller)*
- ➔ *SixGame has instances of Point and Player objects*
- ➔ *SixGameView has instances of PointLabel objects (display for Point objects)*

Updated



- ➔ *MainStartUp uses the JPanel and ActionListener modules*
- ➔ *SixGameView uses the JPanel, ActionListener and MouseListener modules*
- ➔ *PointLabel uses the JLabel module*
- ➔ *SixMensMorris class creates an instance of MainStartUp, to create the window that starts the game*
- ➔ *MainStartUp creates the SixGame module, and SixGame creates the SixGameView module*
- ➔ *SixGame manipulates SixGameView, which displays SixGame's information (SixGame is model, SixGameView is view/controller)*
- ➔ *SixGame has instances of Point,Player, and FileIO objects*
- ➔ *SixGameView has instances of PointLabel objects (display for Point objects)*
- ➔ *SixGame has FileIO object for loading a file*
- ➔ *MainStartUp has FileIO object for saving to file*
- ➔ *Player, PointLabel, and Point implement Serializable for file reading/writing*

Traceback to Requirements

R1: An empty 16 spot board setup by user
R2: Two players represented by blue and red discs
R3: Order of play is determined randomly
R4: Allow user to start a new game or enter discs to represent the current state
R5: User can set the current state of a game by selecting the colour and position of a disc
R6: If the user places the discs manually, the system verifies if the positions are legal and will highlight any errors found
S1: Players can make moves in turns
S2: All moves must be legal and verified
S3: Game recognizes when a player has won. Result of game is displayed at all times
S4: Players can store an unfinished game and re-start a stored game
T1: Choose either 2 player mode or 1 player against computer mode
T2: Rules should be identical against the computer
T3: Order of play is random against the computer
T4: Computer AI makes reasonable and legal moves

Requirements	**Modules**
R1	SixMensMorris, SixGameView, SixGame
R2	Player, Point, PointLabel
R3	SixGame
R4	MainStartUp
R5	SixGameView, Point, PointLabel
R6	SixGameView
S1	SixGame, SixGameView
S2	SixGameView, PointLabel, Point
S3	SixGameView
S4	MainStartUp
T1	MainStartUp
T2	SixGame, SixGameView, PointLabel, Point
T3	SixGame
T4	SixGame

Internal Review/Evaluation

First Milestone:

Our program meets the requirement specified in the outline and is able to create the game board needed for Six Men's Morris. Players have the option to either start a new game or change the state of the current one by positioning the discs (the program will error check the positioning to make sure it is legal and possible). The next stage will require the implementation of victory conditions and thus new modules may need to be introduced to create the logic. Modifications to current classes and methods might also be needed so that multiple tasks could be performed or to increase efficiency.

Second Milestone:

We are very pleased with the new updated revision of our program as it meets all the design requirements and is functioning properly. The user interface has been greatly enhanced, displaying the current state of the game (e.g if a game is in progress, if a player has won, and etc.) and fully allows two players to play and complete a game with error verification and logic checks. In addition to the display improvements, the coding was also polished as more methods were added to increase modularity.

Additional improvements to the design for the future could include adding the ability to restart in the middle of a game. In addition, the potential to continue the previous game in progress if the program was closed prematurely could also be explored. Finally, the user interface can be refined even further by adding counters for number of pieces and for score tracking.

Third Milestone:

For the third revision of our game, we were able to satisfy the design requirements by introducing a computer AI into the game. Players are now able to play with either another person, or versus the programmed computer AI. In addition, a save/load feature has been implemented for the user's convenience which allows the player to save and load a game without manually inputting the disk positions.

Overall, we are very satisfied with the final product. It was able to perform all the necessary conditions while maintaining a unique interface that separates it from other similar programs. If our program were to be updated in the future, possible upgrades would be to improve visual fidelity of the game, and to add more AI algorithms to accommodate various skill levels.

Program Test Report

Test #1: The game starts
Test Type: Black Box

Case	Expected	Actual	Pass or fail
Click on the new game button	A new window with a new game starts	When the button is clicked, the new window is opened with the game	Pass

Test #2: Once started, the player should be able to make a move (place a disk)
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
Click on a spot when blue turn is stated	The spot gets a blue disk	Blue disk appears	Pass
Click on a spot when red turn is stated	The spot gets a red disk	Red disk appears	Pass

Test #3: Once the first player places their disk, it becomes the opposite player's turn
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
First player is blue and does their turn	Becomes red player's turn	Red player's turn stated top right	Pass
First player is red and does their turn	Becomes blue player's turn	Blue player's turn stated top right	Pass

Test #4: Player should not be able to place disk on spot that already has a disk (legal moves only)
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
Red player attempts to place their disk on a blue spot	Turn is not allowed and stays as red player's turn until a valid move is performed	Turn does not go through, stays as red player's turn	Pass

Blue player attempts to place their disk on a blue spot	Turn is not allowed and stays as blue player's turn until a valid move is performed	Turn does not go through, stays as blue player's turn	Pass
Red player attempts to place their disk on a red spot	Turn is not allowed and stays as red player's turn until a valid move is performed	Turn does not go through, stays as red player's turn	Pass
Blue player attempts to place their disk on a red spot	Turn is not allowed and stays as red player's turn until a valid move is performed	Turn does not go through, stays as blue player's turn	Pass
Blue player attempts an illegal diagonal move	Turn is not allowed and stays as blue player's turn until a valid move is performed	Turn does not go through, stays as blue player's turn	Pass
Red player attempts an illegal move (not an adjacent spot movement)	Turn is not allowed and stays as red player's turn until a valid move is performed	Turn does not go through, stays as red player's turn	Pass

Test #5: Game recognizes when a player has won. Result of game is displayed at all times

Test Type: Black Box

Case	Expected	Actual	Pass or Fail
Red player wins (reduced blue disks on board to below 3)	Program recognizes red as winner when red has less than two disks left. Shows red as winner	Option box appears recognizing Red as winner	Pass
Blue player wins (reduced red disks on board to below 3)	Program recognizes blue as winner when red has less than two disks left. Shows blue as winner	Option box appears recognizing Blue as winner	Pass

Test #6: Players can store an unfinished game and re-start a stored game
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
Save option is available to player	Can save current game state to a filename of choice (using dialog box)	Dialog box appears and file is saved as intended	Pass
Load option is available to player	Can load a game state from a chosen file (button in home screen) and continue playing	Dialog box appears and file is loaded as intended, game can be continued	Pass

Test #7: User can create a game state and load it
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
User chooses preset game option	Window which allows user to create prese game appears	Window appears as expected	Pass
Load custom game	User is able to load their preset game	Preset game is loaded	Pass

Test #8: User places disks incorrectly on preset board
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
User tries to place more than 6 red disks on board	Error message showing incorrect number of red pieces appears	Error window appears as expected and board state resets	Pass
User tries to place more than 6 blue disks on board	Error message showing incorrect number of blue pieces appears	Error window appears as expected and board state resets	Pass
User places 2 red and no blue disks on board	Error message showing incorrect number of blue and incorrect number of red pieces appears	Error window appears as expected and board state resets	Pass

Test #9: Option to play against AI is available and working
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
Can the user initiate a Vs AI game	Opening window has button Vs AI available to the user	The button is at opening window	Pass
Vs AI window loads	Clicking Vs AI button opens a new window with the game	User selecting the button successfully loads a new window with a new game	Pass

Test #10: User can place disks for their turn (AI game)
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
User gets first turn and it is stated at top right	Users turn at initialization	It is Blue player's turn (user)	Pass
After initializing game, player can place a disk	Disk can be placed upon clicking an empty spot	Disk was placed	Pass
Rules for two player remain the same for AI	Player cannot move enemy disks or move to non-adjacent spots	Player’s disks are restricted as expected	Pass

Test #11: AI can place disks after the users turn
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
AI gets a turn after the user	Red's turn after user places disk	Red gets a turn	Pass
AI actually places a disk when it is its turn	Red disk gets places on red turn	Red disk is placed	Pass
Disks are placed in different spots every game (no pre determined set of turns)	Red disks don't get placed in the exact same spots every game	Red disks don't seem to follow predetermined turns	Pass

Test #12: AI can mill pieces and win game
Test Type: Black Box

Case	Expected	Actual	Pass or Fail
AI mills after getting 3 disks in a row	Blue disk is removed after Red gets 3 disks in a row	Blue disk is removed as expected	Pass
AI can win the game (AI not complex, so testing this takes longer)	Blue disks go below 3, Red wins the game (dialog appears)	Dialog appears to show that Red won, then returned to main menu	Pass