Six Men's Morris Documentation

Benjamin D. Miller - 001416516
Matthew Kipp - 001303604
William Tran - 001407613
COMP SCI 2ME3 / SFWR ENG 2AA4

April 8th, 2016

ALL CHANGES FOR ASSIGNMENT 3 ARE HIGHLIGHTED IN GRAY

Classes and Modules

The classes have been separated into an MVC format in order to improve efficiency by separating the workload. The *View* package contain the main GUI class, VIew, as well as a class for an additional GUI element. The *Controller* package are event handlers for each different button that must interact with the *Model*. The *Model* package contains the Node class which hold information on each node and the Data class which has the remaining information on the board.

Model Classes:

- Node - Contains position and current colour of each node on the board
- Data - Holds access to all nodes on the board as well as the turn order and methods to alter the data according to the state of the game
- GameState - An enum data type to determine state that the game is in

Controller Classes:

- NodeHandler - Event handler for the node buttons. Allows user to place, move, and remove pieces according to the state of the game.
- ColourHandler - Event handler for colour buttons. Changes the turn to the colour pressed. Sandbox only.
- ResetHandler - Event handler for the reset button. Resets all nodes to black and updates the board. Sandbox only.
- CheckHandler - Event handler for the check button. Determines if the current state of the board is valid or not. Sandbox only.
- AIHandler - Handles AI movement. Determines current state of board and moves AI colour based on state.

View Classes:

- View - The GUI of the game. Allows the game to be seen and played.
- ConfirmBox - A GUI element to return a boolean value for a Y/N player decision

Model Classes

Class: Node

Package: Model

Contains layer and index coordinates and colour of a node corresponding to a node on the game board.

Interface:

Uses:

None

Access Programs:

Node(l:int, p:int, col:String)

- Constructs a new Node object using the given layer, l, index, i, and colour, col.

getLayer():int

- Returns the layer of the node

getPosition():int

- Returns the position of the node

getColour():String

- Returns the colour of the node

setColour(col:String)

- Sets the colour of the Node to col

isConnected(other:Node):boolean

- Returns true if Node is connected with 'other' Node

Class: Data

Package: Model

The data class uses the data and checks whether or not a valid board has been played

Interface

Uses:

Node, GameState

Access Programs:

getNumPieces():int

- Returns number of Nodes on the board

getBlueCount():int
- Returns number of blue Nodes on the board

getRedCount():int
- Returns number of red Nodes on the board

getColour(layer:int,index:int):String
- Return the colour of a node at the specified layer and index

setColour(layer:int,index:int)
- Sets the colour of the node at the specified layer and index according to which player's turn it is

reset()
- Resets all node colours to black and picks a random player's turn
- Change piece counters to zero and change GameState to placement phase

getTurn():boolean
- Return true if it is blue turn

changeTurn:None
- Change the turn to the other player

changeTurn(col:String)
- Change the turn to the specified colour, col.

chooseTurn()
- Randomly select the turn order

setState(nextState:GamState)
- Sets current state of the game to enum type

getState():GamState
- Returns current state of the game as enum type

setMove(layer:int, index:int)
- Determines which node is the movingNode

canMove(layer:int, index:int):boolean
- Returns true if the movingNode is connected to the layer and index given

swapNode(layer:int, index:int)

- Swaps the movingNode's position and colour with a "black" node's position and colour
- clears the movingNode variable

checkWin()

- Checks if a player has won if:
    - A player has 2 pieces remaining or
    - A player can not make any legal moves

singleTriple(layer:int, index:int):boolean

- Returns true if a given layer and index is part of a triple on the board

mill(layer:int, index:int)

- Changes colour of given layer and index to black
- Reduces that colour's count by one

hasMove():boolean

- Returns true if moveNode is not null

save()

- Saves game to savefile.txt as CSV's containing:
    - Current game state, outer ring of Nodes, inner ring of Nodes

load()

- Loads game state, outer ring and inner ring of Nodes from savefile.txt

setAI()

- Creates AIHandler object, sets isAIOn to true

moveAI()

- Tells AIHandler to conduct an AI move

Class: GameState

Package: Model

An enum data type to keep track of current phase of the game

Interface:

Uses:

None

Access Programs:

GameState():enum

- Contains state of the game

Controller Classes

Class: NodeHandler

Package: Controller

Change colour of Nodes on the board

Interface:

Uses:

Data, View, GameState

Access Programs:

NodeHandler(layer:int,index:int)

- newly created nodes have their own layer and index

handle(event:Event)

- Determines phase of GameState and has node act accordingly:
  - SANDBOX: colour of node is changed to current colour
  - PLACEMENT: colour of node is changed to current colour if currently “black”
  - MOVEMENT: swaps coloured node with “black” node if isConnected
  - MILL: removes an opposite coloured piece
- Updates View

Class: ColourHandler

Package: Controller

Change active colour to change Nodes to

Interface:

Uses:

Data

Access Programs:

ColourHandler(colour:String)

- assign selected colour to Handler

handle(event:Event)

- change current active colour and updates view

Class: ResetHandler

Package: Controller
Clear the board of placed Nodes

Interface:

Uses:

Data, View

Access Programs:

handle(event:Event)

- reset board and update view

Class: CheckHandler

Package: Controller
Confirm a legal board exists

Interface:

Uses:

Data

Access Programs:

handle(event:Event)

- count number of red and blue nodes on board
- determine if there is proper amount of red and blue on board

Class: AIHandler

Package: Controller
Calculate AI moves for one player mode

Interface:

Uses:

Data, View, GameState

Access Programs:

AIHandler(name:Type)

- Constructs AIHandler object

makeMove(currentTurn:boolean)
- Performs AI turn if currentTurn is true
- Decides on move logic by checking GameState

View Classes

Class: View

Package: View

The view class provides the GUI portion of the program. It allows the user to switch between scenes, and gives the user the ability to see what the program is accomplishing.

Interface:

Uses:

Data, NodeHandler, CheckHandler, ResetHandler, ColourHandler, ConfirmBox, GameState

Access Programs:

main(args:String[])
- Launches the application

start(primaryStage:Stage)
- Sets up the GUI elements

update()
- Update colour of all nodes
- Update turn indicator
- Update game state label

Class: ConfirmBox

Package: View

The ConfirmBox class is necessary to create a confirm box dialogue when the user attempts to exit the program. This then runs the necessary exit code and

Interface:

Uses:

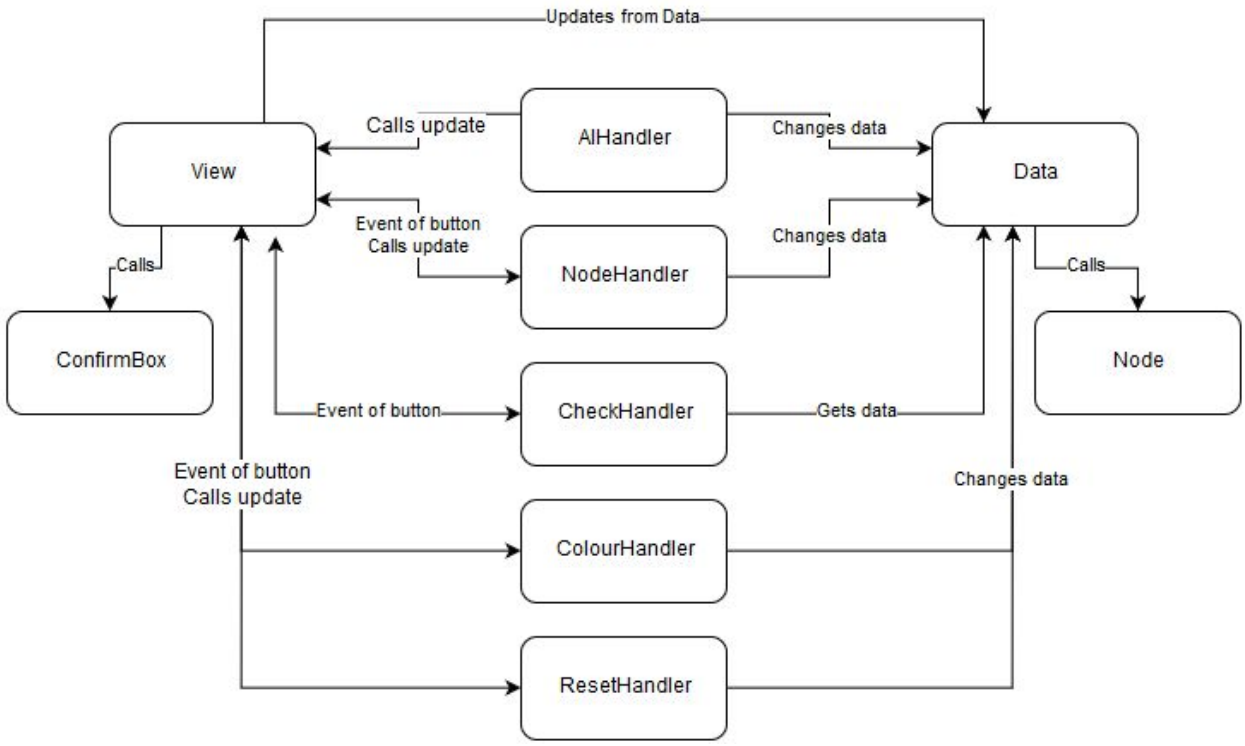
View

Access Programs:

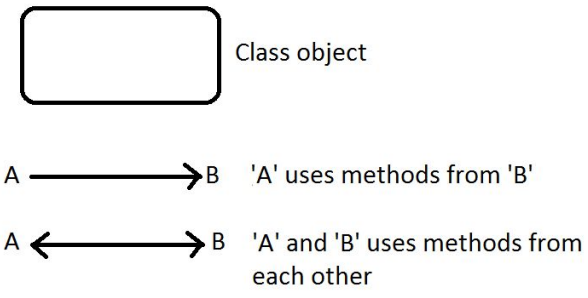
display(title:String, message:String):boolean

- display popup with the title
- display message on the popup
- ask Y/N and return result

Uses Relationship



LEGEND:



Requirements

R1) Make moves in turn
R2) Determine a win state
R3) Randomly determine the starting player
R4) Display gamestate at all times
R5) Start new game, save game, load game
R6) User is able to start a new game, save a game, or load a game
R7) Detect a mill and allow the user to remove an opposing piece
R8) AI tries to complete own mills and block opponent mills

Req.	**Module**
R1	NodeHandler, Data
R2	Data
R3	Data
R4	Data, View
R5	Data, View
R6	NodeHandler, Data
R7	NodeHandler, Data
R8	AIHandler, Data

Requirements are satisfied by their corresponding modules on the right hand of the table.

Private Entities:

Class Node:

Private Variables:
int layer
- final variable to hold which layer of nodes on the board

int index
- final variable to hold position on the layer

String colour
- variable to hold the colour of the node

Class Data

Private Variables:
int TOTAL_LAYERS
- number of layers on the board

int TOTAL_POSITIONS
- number of positions on each layer

Node[][] nodes
- 3D array to hold all nodes in the board

boolean isBlueTurn
- true if it is blue's turn, false if red

boolean isAIOn
- true if AI is playing

AIHandler AIBot
- Contains AI object, null if not playing

Private Methods:
void chooseTurn()
- run at the start of a game
- determines which player starts

Class CheckHandler

Private Variables:

int redCount

- holds amount of red Nodes on the board when checking for a legal board

blueCount:int

- holds amount of blue Nodes on the board when checking for a legal board

Class ColourHandler

Private Variables:

String colour

- holds which colour the button corresponds to

Class NodeHandler

Private Variables:

int layer

- final variable to hold which layer of nodes on the board

int index

- final variable to hold position on the layer

Class AIHandler

Private Variables:

boolean hasBlueTurn

- Colour of AI's piece

Private Methods:

Node findRedPair(layer:int, index:int)

- Finds pair of opponent pieces
- Returns opponent piece if removable, otherwise null

Node findOpenTriple(col:String)

- Finds pair of a colour
- Returns Node if pair can be milled by filling in a black Node

int checkAdj(layer:int, triple:int, index:int)

- Check for adjacent blue piece
- Returns index of blue piece, otherwise -1

Node ableToMove(layer:int, index:int)

- Check for adjacent black Node

- Return Node if there is a black Node, otherwise null

Node findMill()
- Tries to find opponent mill to remove
- Choose random red node otherwise
- Returns Node to be removed

Node[] findMove()
- Determines a piece to move
- Returns two Nodes, first is the empty Node and then the blue Node to move

Node findPlacement()
- Determines a spot to place a piece
- Prioritizes finishing a mill and blocking a mill
- Returns Node to be placed

void millPhase()
- Removes opponent piece during mill

void placementPhase()
- Places piece during placement phase

Class View

Private Variables:
int WIDTH, int HEIGHT
- final variables for application dimensions

int TOTAL_LAYERS
- number of layers on the board

int TOTAL_POSITIONS
- number of positions on each layer

Stage window, Scene mainScene, Scene boardScene, Scene creditScene
- gui elements for the application and scenes

Button onePlayer, twoPlayer, credits, exitButton, quitButton, return Button
- buttons for scene navigatioin

Button reset
- button to reset the state of the board/reset the game

Button check

- button to validate the state of the game

Button red, blue

- buttons to forcefully change the player turn

Button[][] nodes

- 3D array of buttons representing each node in the game

String blackNode, redNode, blueNode

- contains css style for node images

String redHighlight, blueHighlight

- contains css style for highlighted node images
- to show who's turn it is

String boardImage

- contains css style for board image

Label label1,curState

- Provides the labels for the current state during gameplay and the title text

Private Methods:

void closeProgram()

- call exit confirmation
- exit if true

GridPane addGrid()

- create a gridpane, positioning all node buttons
- returns a GridPane to position on the application

Tabular Expression of the GameState FMS

CURRENT STATE	CONDITIONS					RESULT
PLACEMENT	click coloured node					NO CHANGE
	click black node	numPieces < 11	singleTriple = true			ownColour++, numpieces++, MILL state
			singleTriple = false			ownColour++, numpieces++, changeTurn
		numPieces = 11	singleTriple = true			ownColour++, numpieces++, MILL state
			singleTriple = false			ownColour++, numpieces++, changeTurn, MOVEMENT state
MOVEMENT	click black or opp node					NO CHANGE
	hasMove == false	click own node				moveNode = clicked node
	hasMove == true	click opp node				NO CHANGE
		click own node				moveNode = clicked node
		click black node	canMove = false			NO CHANGE
		canMove = true		singleTriple = true	checkWin = true	swapNode, changeTurn, WIN state
					checkWin = false	swapNode, Mill state
				singleTriple = false	checkWin = true	swapNode, changeTurn, WIN state
					checkWin = false	swapNode, changeTurn
MILL	click own node or black node					NO CHANGE
	click opp node	singleTriple == true && oppCount >= 3				NO CHANGE
		singleTriple == false || oppCount == 3	numPieces < 12			oppColour--, PLACEMENT state
			numPieces == 12	checkWin = true		oppColour--, changeTurn, WIN state
				checkWin = false		oppColour--, changeTurn, MOVEMENT state
WIN	Every button except quit					NO CHANGE

Preconditions and Postconditions for Game States

Game State	Preconditions	Postconditions
SANDBOX	The GUI must be in the main menu - then sandbox mode must be selected sandboxPhase() == True	No set postconditions - once sandbox mode has finished the game will be back in the menu
PLACEMENT	No pieces must be on the board and a new game must have begun numPieces < 12 placementPhase() == True	All twelve pieces will have been placed on the board
MOVEMENT	All twelve pieces must have been either placed or removed Data.getNumPieces() == 12 blueCount > 2 redCount > 2 movementPhase == True	Pieces are able to move
MILL	A player must have three pieces in a row Data.singleTriple() == True EndGame == False	The player removes one of the opponent's pieces
ENDGAME	The win state must have been fulfilled and one of the players should be in the win state movementPhase == False EndGame == True	After the end game state no pieces will be able to be moved or placed until a new game is started

Private entities are maintained by using ‘private’ key word in the Java, and can only be changed within the class that contains them. Few private variables are made available to external classes by using accessor functions (getters and setters) to permit access to them.

Design Review

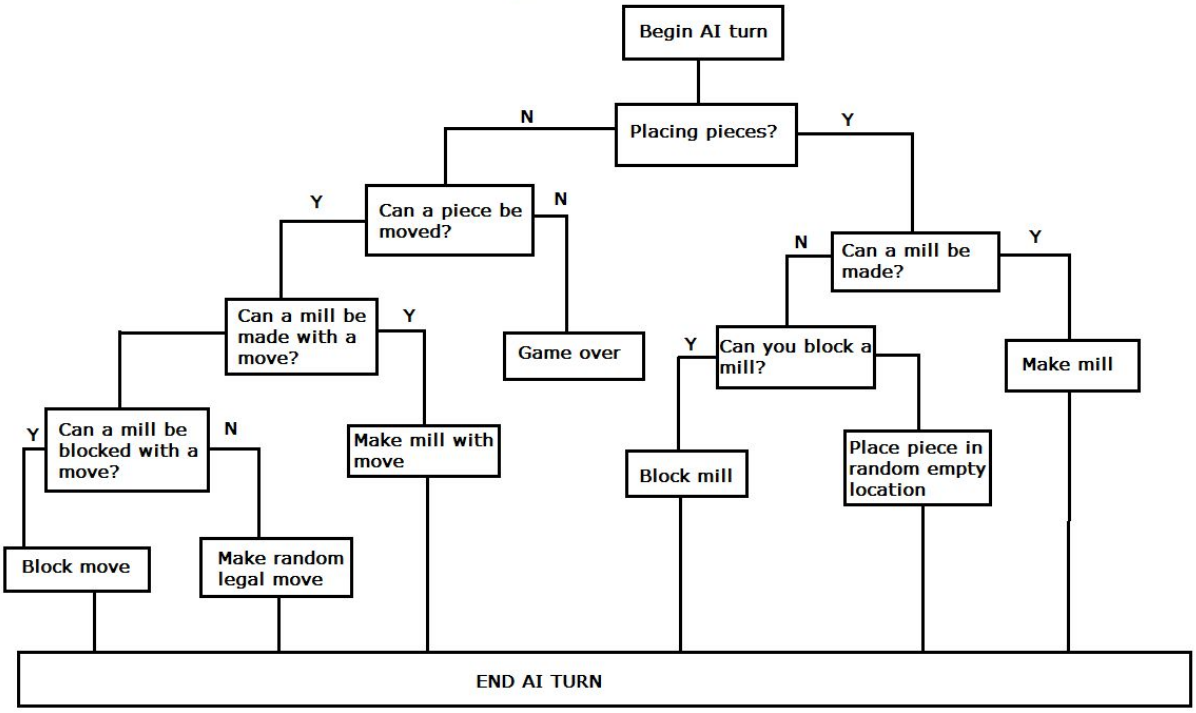
The design has expanded on the two player version of 6 Men’s Morris by adding the option of playing against the computer. The AI was created with three priorities in mind. The first is to complete it’s own mill, the second is obstructing the opponent from making a mill, and the third is to simply make a valid move. Further improvement could have been made on the AI by adding a priority to move a piece in and out of moves as well as predicting moves in advance.

More information could be shown visually such as the number of pieces left to play or which piece has been selected to move. These could be added to show the exact state of the board and remove confusion. Additionally, the AI should have a delay so the player has time to process how the AI has moved. This could also be achieved by giving the AI animations for when it makes a move.

One point that could be changed is to split the MOVEMENT state into 2; the first being selecting the piece to move and the second being selecting the location to move the piece to. However we kept this as one state as that is more a more intuitive way to think of the action. Due to this, loading the game will not keep the selected piece to move.

Another thing that would be good to change is to add more message prompts or information labels over the course of the game. Currently, the only print is the state of the game. What we would like to add is prompts when the user tries to make an invalid move.

AI Placement and Movement Logic: White-Box Testing



Test Report Document

Tested by:	Ben M, Matthew K., William T.
Tested on:	April 7th, 2016
Total number of test cases:	**15**
Number of test cases passed:	**14**
Number of test cases failed:	**1**

Test Case ID	Test Cases Checked	Input	Output	Status
TEST_01	2 Player mode, Check if both players can play	Place pieces on board, select player 2 game	Game successfully finishes	**PASSED**
TEST_02	Check save and load game	Place piece on board, quit game and reload. Then quit program and reload	Game successfully reloads to the previous board. Game state stored in txt file	**PASSED**
TEST_03	Place a piece on top of another piece	Place a piece, then attempt placing another on top of existing	Could not replace already placed piece. Which is the desired output	**PASSED**
TEST_04	Exiting the game	Press exit button	Exits the game successfully after warning message	**PASSED**
TEST_05	Sandbox mode	Test sandbox mode and experiment within it	Sandbox mode works as it is supposed to, freely allowing piece placement	**PASSED**
TEST_06	Post win trying to place/move pieces	Tried moving and placing pieces after game has been won	Unable to do so, just as intended to	**PASSED**
TEST_07	Reload game without saving, game should only save if the user chooses to save the game	Saved current state of game, created a new instance and then clicked load game	The game loaded was the game that was saved, only saves game after “save game” button has been pressed	**PASSED**

TEST_08	Ensure game starts on both blue and red randomly	Simulated 30 new games. Red started 12 times, Blue started 18 times	Since both colours have the ability to start relatively evenly the case passes	PASSED
TEST_09	Ensure the computer can play as both red and blue	Simulated 30 new games. Computer was blue 30 times, and red 0 times.	The computer player (AI) is blue 100% of the time, not what was required	FAILED
TEST_10	Place pieces	Placed a piece on each position after several games	Was successfully able to place and view pieces on the game board	PASSED
TEST_11	Three in a row placement, checks that the player should remove opponent's piece	Placed three colours of blue and red separately	After both cases the GUI prompts the user to	PASSED
TEST_12	AI testing, checks computer places game if they go first	To test started a new game until the computer was randomly selected to go first	If the computer begins the game their piece is automatically placed, if not the AI waits	PASSED
TEST_13	Checks if the AI chooses to place a piece creating a triple	To test simulated games where the AI would have the opportunity to place a triple	AI successfully completes the triple	PASSED
TEST_14	Checks that the AI attempts to block a triple place by the user	To test simulated a game where the AI would have the opportunity	AI successfully blocks the triple where it's own triple is not possible	PASSED

		to block a triple		
TEST_15	Checks that the AI for moving pieces is working and operating correctly	To test, simulated placing a pieces and then continued the game with moving	The gameplay continues where the AI moves their pieces attempting to make and block triples	**PASSED**

Summary of Test Report

After the test cases have been checked and passed, it has been determined that there are no flaws or visible bugs within Six Men's Morris at the time this document was published (04/7/16).

Tests were done on the following areas: gameplay, menu selection, game modes, and AI.