Six Men's Morris Documentation

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

February 22nd, 2016

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

Controller Classes:

- NodeHandler - Event handler for the node buttons. Changes the colour of the attached node button.
- ColourHandler - Event handler for colour buttons. Changes the turn to the colour pressed.
- ResetHandler - Event handler for the reset button. Resets all nodes to black and updates the board.
- CheckHandler - Event handler for the check button. Determines if the current state of the board is valid or not.

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.

Uses:

None

Access Programs:

void Node(int l,int i,String col)

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

int getLayer()

- Returns the layer of the node

int getIndex()

- Returns the index of the node

String getColour()

- Returns the colour of the node

setColour(String col)

- Sets the colour of the Node to col

Class: Data

Package: Model

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

Uses:

Node

Access Programs:

String getColour(int layer,int index)

- Return the colour of a node at the specified layer and index

void setColour(int layer,int index)

- Sets the colour of the node at the specified layer and index according to which player's turn it is

void reset()

- Resets all node colours to black and picks a random player's turn

boolean getTurn()

- Return true if it is blue turn

changeTurn:None

- Change the turn to the other player

void changeTurn(String col)

- Change the turn to the specified colour, col.

void chooseTurn()

- Randomly select the turn order

Controller Classes

Class: NodeHandler

Package: Controller
Change colour of Nodes on the board

Uses:

Data, View

Access Programs

NodeHandler(int layer,int index)

- newly created nodes have their own layer and index

void handle(Event)

- change colour of node to current active colour and update view

Class: ColourHandler

Package: Controller
Change active colour to change Nodes to

Uses:

Data

Access Programs

ColourHandler(String colour)

- assign selected colour to Handler

void handle(Event)

- change current active colour

Class: ResetHandler

Package: Controller
Clear the board of placed Nodes

Uses:

Data, View

Access Programs:

void handle(Event)

- Reset board and update view

Class: CheckHandler

Package: Controller
Confirm a legal board exists

Uses:

Data, View

Access Programs:

void handle(Event)

- Count number of red and blue nodes on board
- Determine if there is proper amount of red and blue on board

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 JavaFX library to assist in the visualization of the game.

Uses:

Data, NodeHandler, CheckHandler, ResetHandler, ColourHandler, ConfirmBox

Access Programs:

void main(String[] args)

- Launch the application

void start(Stage primaryStage)

- Setup all GUI elements

void update()

- Update colour of all nodes
- Update turn indicator

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:

Similar to the View class, ConfirmBox uses the JavaFX library to assist in visualization.

Uses:

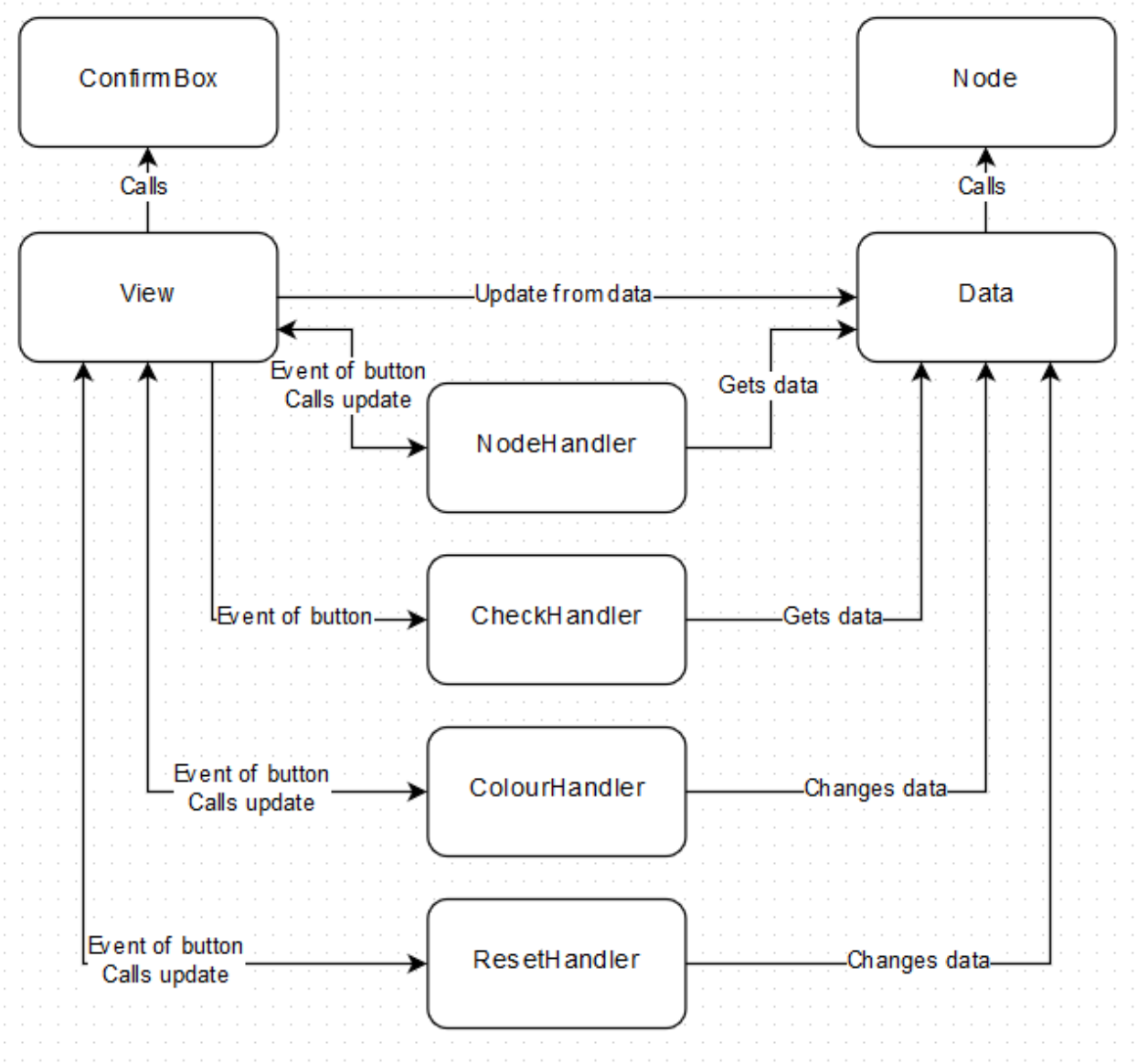
View

Access Programs:

boolean display(String title, String message)

- Display popup with the title
- Display message on the popup
- Asks Y/N and returns boolean result

Six Men's Morris Relationship Flow Chart



Requirements

R1) Click red button to change current team colour to red
R2) Click blue button to change current team colour to blue
R3) Click a Node on the board to change colour of selected Node to current team colour
R4) Check button to confirm board contains legal placement of coloured Nodes
R5) Reset button to return board to its initial state

Req.	**Module**
R1	ColourHandler, Data
R2	ColourHandler, Data
R3	NodeHandler, View, Data, Node
R4	CheckHandler, Data
R5	ResetHandler, View, Data

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

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 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
- Shows who's turn it is

String boardImage

- Contains CSS style for board image

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

Design Review

The design is implemented in an MVC format. It was decided that all **Controller** classes will be event handlers, even if there is only one instance of the handler used. This was to maintain the principle of high cohesion by keeping the **View** class strictly about the GUI. However, MVC was not strictly followed as the **View** class looks at information from the Data class directly and not through the **Model** classes. The reason why we chose to design this way is to keep the **Model** classes from changing GUI elements as any changes that occur are done in the **View** class.

For a later revision, the alert dialogues in the checkHandler class should be moved to the ConfirmBox class for reusability.

Testing

Button Sequence	Expected Output	Works?
Blue -> Red	Red button highlighted	true
Red -> Node(1,1)	Node(1,1) turns red	true
Red -> Blue -> Node (1,1)	Node(1,1) turns blue	true
Check	Prints valid board	true
Reset	Removes nodes from board	true
Red -> Node(1,1) -> Reset	Removes nodes from board	true
Red -> Node(1,1) -> Blue -> Node(1,1)	Node(1,1) remains red	true
Blue -> Node(1,1) -> Node(1,2) -> Check	Invalid board	true
Blue -> Node(1,1) -> Node(1,2) -> Node(1,3) -> Node(1,4) -> Node(1,5) -> Node(1,6) -> Node(1,7) -> Check	Invalid board, more than 6 pieces of the same colour	true

After several tests and all test cases have been checked, it has been determined that there are no flaws or visible bugs at the time this document is published.