|  |  |
| --- | --- |
| Program #4 | Planning Document  James Scott  Colin Riley  Stephen Belden  Shaya Wolf  Neil Carrico  04/22/2016 |

**Project 4**

The first thing we did for this project was randomize the tiles. We found that randomizing the location was much easier than rotating the orientation. To instantiate this change, we determined that our setup function needed to be called sometimes with shuffling the tiles and sometimes without, so we added a parameter to this called Boolean shuffle. Therefore, we can now call setup(true) if we want to set up the game board with shuffling the tiles and setup(false) to do the opposite.

We used this same concept to randomize the orientation of the tiles. Since the tiles orientation never needs to be randomized without the location being randomized and the location being randomized never needs to happen without randomizing the orientation, we were able to keep the one additional parameter and use it for both, rather than use two parameters.

Then we added a right-click function to our game. When the user right-clicks on a tile, that tile is rotated 90 degrees clockwise. The hardest part of this was making the rotation “sticky.”

// Include in the paragraph above how we made the rotations sticky.

We then changed the functionality in our newGame and reset buttons. NewGame now produces a new game with shuffled tiles. The tiles in a new game are also oriented randomly. The reset button returns all of the pieces to the randomized locations that they started in before the user moved and rotated anything.

Of course, our program starts and exception handles in the case that data can’t be read from default.mze. Then, we draw the lines on the tiles, which are randomly placed in the side panels on the board. The orientation is also randomized. Every time the use starts a new game, the initial layout is changed. We have also implemented the following

1. No two tiles can occupy the same space – this was done in the original planning stages when we decided not to have spaces in the first place and to simply make everything a tile.
2. Rotations are sticky – // reiterate briefly from above
3. The tiles can be removed from the game board – this is done with the same action that places them on the game board. Also, it doesn’t matter what order you click the tiles or game board in. The tiles continue to swap seamlessly.

Our Buttons are named exactly New Game, Reset, and Quit. These are the only three buttons and they appear in the same spot on the screen as the picture in the assignment document.

**Future Plans**

We still need enforce legal/illegal moves and win conditions. We will have to do plenty of testing on the game code given that more and more cases are beginning to build and that is inflating the test cases rather quickly.

Meeting times continue to fluctuate based on class schedules and class presentations. Most work is being done throughout the week separately, and then being combined throughout the week, utilizing Slack. We will continue to meet Friday’s to finish projects before they are due.

**Anonymous Class Explanation**

We were instantiating new action listeners in the actionPerformed() method, which required the use of anonymous classes. This was a problem, so we removed these classes by deleting them and instead called the getActionCommand() method. This was definitely the more proper and clean than what we had. So we will be keeping these changes and avoiding anonymous classes in the future.

**Screen Resolution Explanation**

Rather than rebuild the code back up using a different screen dimension, we found 3 lines of code that we include if we are working on a lower resolution machine and comment out on a higher resolution machine. This code can be found in the centerTiles() method under the // Computer Screen comment.

We found that this type of change could have been made at the beginning of the project and the game board would have fit any screen size. In future software development, we will take screen resolution into account and set our parameters using Java utilities that return screen dimensions rather than hardcoded integers.

**UML**

<<Interface>>  
Action Listener

GameWindow

+ << constructor>>GameWindow

+ actionPerformed (actionevent)

+ newGame()

+ reset()

+ setup(Boolean)

+ emptyRow(GridBagConstraints,

int, int, Tile[])

+ sidePanels(GridBagConstraints,

int, int, Tile[])

+ centerTiles(GridBagConstraints,

int, int)

+ addButtons(GridBagConstraints)

+ shuffleArray(Tile[])

+ setLeftClicked(Tile)

+ setRightClicked(Tile)

+ readInt(FileInputStream)

+ convertToInt(byte[])

+ convertToFloat(byte[])

+ newButton: JButton

+ resetButton: JButton

+ quitButton: JButton

+ lastClicked: Tile

Line

+ << constructor>> Line(Point, Point)

+ getBegin(): Point

+ getEnd(): Point

+ debugPrint()

- begin: Point

- end: Point

**Uses**

**Uses**

Tile

+ << constructor >> Tile(int, Line[])

+ << constructor >> Tile(int)

+ getID()

+ setID(int)

+ getOrient(): int

+ setOrient(int)

+ incOrient()

+ getStart\_orient(): int

+ setStart\_orient(int)

+ getStart\_Loc(): int

+ setStart\_Loc(int)

+ getLines(): Line[]

+ setLines(Line[])

+ paintComponent(Graphics)

+ isEmpty(): Boolean

+ makeEmpty()

+ makeLive()

+ switchState()

+ reset():void

+ debugPrint()

+ mousePressed(MouseEvent)

+ mouseClicked(MouseEvent)

+ mouseEntered(MouseEvent)

+ mouseExited(MouseEvent)

+ mouseReleased(MouseEvent)

- ID: int

- lines : Line[]

- isEmpty: Boolean

- orient: int

- start\_orient: int

- start\_loc: int

- border : Border

- NoBorder : Border

JFrame

JLabel

<<Interface>>  
Mouse Listener