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

**Instructions**

* Click a tile to select it. Selected tiles will turn green.
* While a tile is selected, click another tile or a blank spot to move the tile.
* Right-click a tile to rotate it.
* Click “Reset” to move the tiles back to their initial positions.
* Click “New Game” to play with a newly shuffled tile arrangement.
* Click “Quit” to quit. Duh.

**Changes**

The first thing we did for this project was randomize the tiles. We found that randomizing the location was much easier than randomizing the orientation. Sometimes the tiles need to be added to the gameWindow without being shuffled, so we added a Boolean parameter to our setup function called newGame. We can now call setup(true) if we want to set up the game board with shuffling the tiles and setup(false) to reset the game board to its initial status without reshuffling.

We used this same concept to randomize the orientation of the tiles. Since a tile’s 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 use the same parameter for both.

Next we added a right-click function to our game. When the user right-clicks on a tile, that tile is rotated 90 degrees clockwise. We’re rotating the tiles by rotating the graphic2d object associated with the tile.

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. We made sure that there are exactly 4 tiles with each orientation. 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. We draw the lines on the tiles, which are randomly placed in the side panels. Every time the user 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 – That is, tiles keep their orientation when moved.
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.

We changed the color of the boarders on the game board to white. When the boarders were black, it was very difficult to distinguish the boarders from maze lines on the edges of the tiles. The new white boarders make it easy to see when a black maze line rests on the edge of a tile.

**Future Plans**

We still need enforce legal/illegal moves and win conditions. We will have to do plenty of testing on the game code.

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.

**Screen Resolution Explanation**

Rather than rebuild the code back up using a different screen dimension, we wrote 3 lines of code that we include if we are working on machines that don’t have 1000 vertical pixels, which we comment out on a higher resolution machines. 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)

+ << constructor>> Tile(Tile)

+ getID()

+ setID(int)

+ getOrient(): int

+ setOrient(int)

+ incOrient()

+ 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

- border : Border

- NoBorder : Border

JLabel

JFrame

<<Interface>>  
Mouse Listener