|  |  |
| --- | --- |
| Program #5 | Planning Document  James Scott  Colin Riley  Stephen Belden  Shaya Wolf  Neil Carrico  05/02/2016 |

**Instructions**

* Program will still start by loading the default input file.
* Program runs from a command line with directory documentation.
* “New Game” is relabeled “File.”
* File has ONLY “Load” and “Save.”
* If the file is not found, an error message pops up.
* File selector windows allows the user to type in a filename.
* Filenames allow for either full paths or relative paths as well as the current directory.
* Loading a file replaces the currently loaded maze with a new maze.
* If the current maze has been modified, the user has the opportunity to save it before the window closes (both for quit and load).
* If “Reset” is pressed, the game board is reset as if no tiles were moved.
* If the player loads a game on a fresh board, they are not asked to save.
* If the player loads a game or quits a game that is not fresh, they are first asked to save.
* The window layout has not changed.

**Changes**

First, we fixed the button layout to disclude “New Game” and include “File,” which leads to two new buttons, “Load” and “Save.” Then, we added functionality to the “Load” and “Save” buttons. The Load button loads a previously saved game, which of course means that the save button saves the current game. Loading a maze replaces the current maze.

Both the “Load” and “Save” buttons use a file selector window. This window allows the user to type in a file name. This by extension allows the user to name their saved files as well as search for previously saved games. File names also allow for either full paths or relative paths as well as a current directory.

Then, we added certain logic constraints on these buttons. We included an error message for the case that a user attempts to open a file that cannot be found. A prompt is also included in the event that the currently loaded maze has been modified since the last change and the use selects the “Load” or “Quit” button. This prompt gives the user the opportunity to save their current game. It follows that if the player loads a game on a fresh board, they are not asked to save. Also, if the user tries to save a file as an already used name, an error message pops up.

A filewriter was added in this iteration of the program. We did run into an issue where we were not saving in a file type that entirely met specifications. However, we were able to fix it before this program was due. The filewriter was modified for this program as well, as it had to read in played games.

Other functionality has not changed. The “Reset” button still resets the currently loaded game such that all of the tiles are back to where they were after the load and before any user moves and the “Quit” button still exits the game.

We also explored some options for cleaning our current code. We looked into copy constructors and smoother implementation for the “setLeftClicked()” function. Although there are still current issues with this implementation, it holds promise for Program 6.

**Future Plans**

We still need enforce legal/illegal moves and win conditions. We will have to do plenty of testing of 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. A special Monday meeting was called to adhere to the Monday due date.

**UML**

GameWindow

+ << constructor>>GameWindow

+ actionPerformed (actionevent)

+ file()

+ loadGame()

+ saveGame()

+ reset()

- newWindow()

+ setup(File, boolean, boolean)

+readFromFile(File File)

+fileOutArray() : byte[]

+ emptyRow(GridBagConstraints, int, int, Tile[])

+ sidePanels(GridBagConstraints, int, int, Tile[])

+ centerTiles(GridBagConstraints, int, int)

+ addButtons(GridBagConstraints)

+ shuffleArray(Tile[])

+ setLeftClicked(Tile)

+ setRightClicked(Tile)

+ writeFile(String, byte[])

+ toFullByteArray(byte[],

byte[], int)

+ convertToInt(byte[])

+ convertToFloat(byte[])

+ hexStringToByteArray(String)

- hexString : String

+ fileButton: JButton

+ resetButton: JButton

+ quitButton: JButton

+ saveButton: JButton

+ loadButton: JButton

- lastClicked: Tile

- tiles: Tile[]

- grid: Tile[]

- played: boolean

- basic : GridBagConstraints

<<Interface>>  
Action Listener

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)

+ getLoc()

+ setLoc(int)

+ getID()

+ setID(int)

+ getOrient()

+ setOrient(int)

+ incOrient()

+ getLines()

+ setLines(Line[])

+ paintComponent(Graphics)

+ isEmpty()

+ makeEmpty()

+ makeLive()

+ switchState()

+ reset()

+ debugPrint()

+ mousePressed(MouseEvent)

+ mouseClicked(MouseEvent)

+ mouseEntered(MouseEvent)

+ mouseExited(MouseEvent)

+ mouseReleased(MouseEvent)

- ID: int

- loc: int

- lines : Line[]

- isEmpty: boolean

- orient: int

- border : Border

- noBorder : Border

JLabel

JFrame

<<Interface>>  
Mouse Listener

Main

+ defaultPath : File

+ verbose : Boolean

+ game : GameWindow

+ initialTileState : Tile[]

+initialGridState : Tile[]

+writeTileArray : Tile[]

+ main(String[]) : void

**Uses**