# PROGRAM #6

PLANNING DOCUMENTS AND ANALYSIS

James Scott
Colin Riley
Stephen Belden
Shaya Wolf
Neil Carrico

05/11/2016

**Program 1 | Introduction**

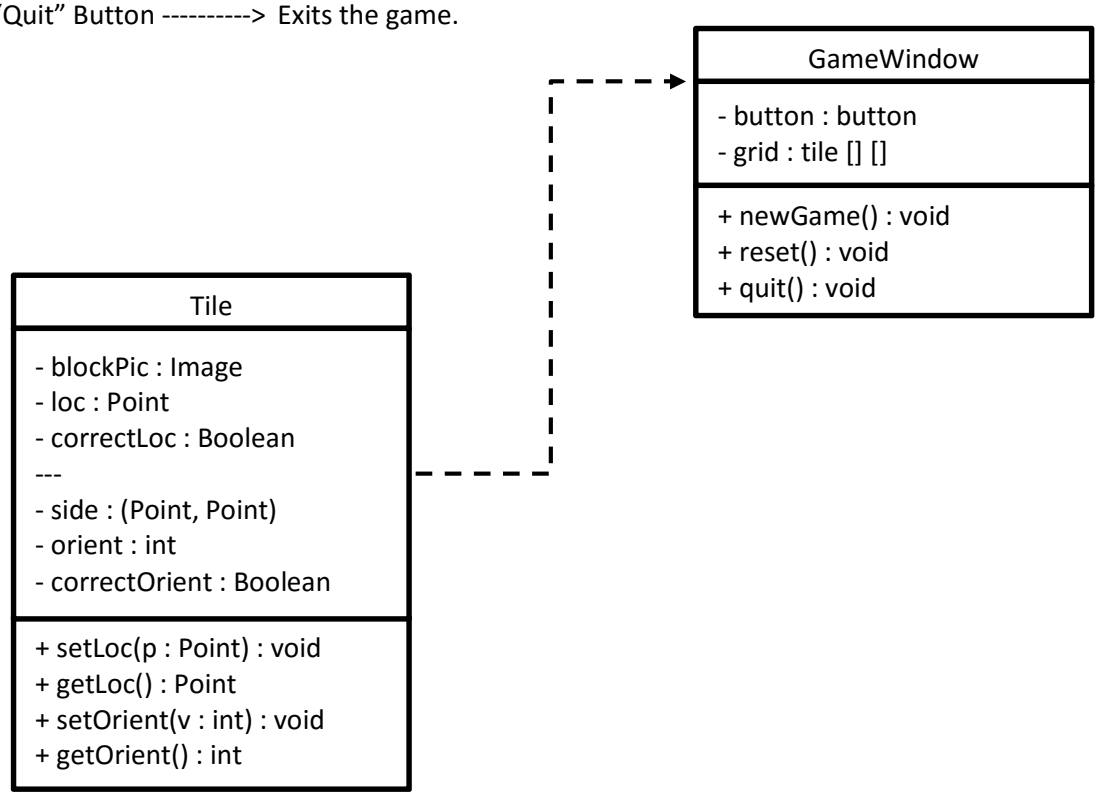
First, we created a UML diagram, consisting of 2 classes. The first class was the tile class. This class implemented the behavior of the tile game pieces. These pieces are squares with sections of a maze image on the top. Each tile has a location and an orientation. Each of these can be accessed and changed using the methods. This functionality has not changed. The second class is the gameWindow. This holds the game board, the 'sidelines,' the buttons and the tiles.

The locations are points on a grid. There are 16 tile locations on the game board. Each tile also has one, and only one, correct space on the game board. This was going to be represented by the Boolean value 'correctLoc' and return true if the tile was in the correct location on the game board. This was instead done in a win conditions function. Each tile also needed to be able to be in any spot on the 'sidelines' and therefore be completely off of the game board as well. If only life was that simple.

The orientation of each tile was originally entered as an integer in degrees in the counterclockwise direction. This was used 'side' in the rotate function. Tiles were instantiated at zero degrees, which is the tile fully in quadrant I. The 'correct orientation' for every tile will be this zero degrees. The other three possibilities are 90 (quadrant II), 180 (quadrant III), and 270 (quadrant IV).

We were able to use the gameWindow code that was provided and create a game window with three interactive buttons, 'sidelines,' and a game board. The three interactive buttons included were:

- "New Game" Button -> Creates a new game with new tiles,
- "Reset" Button --------> Puts the same tiles back at the start position and
- "Quit" Button ----------> Exits the game.



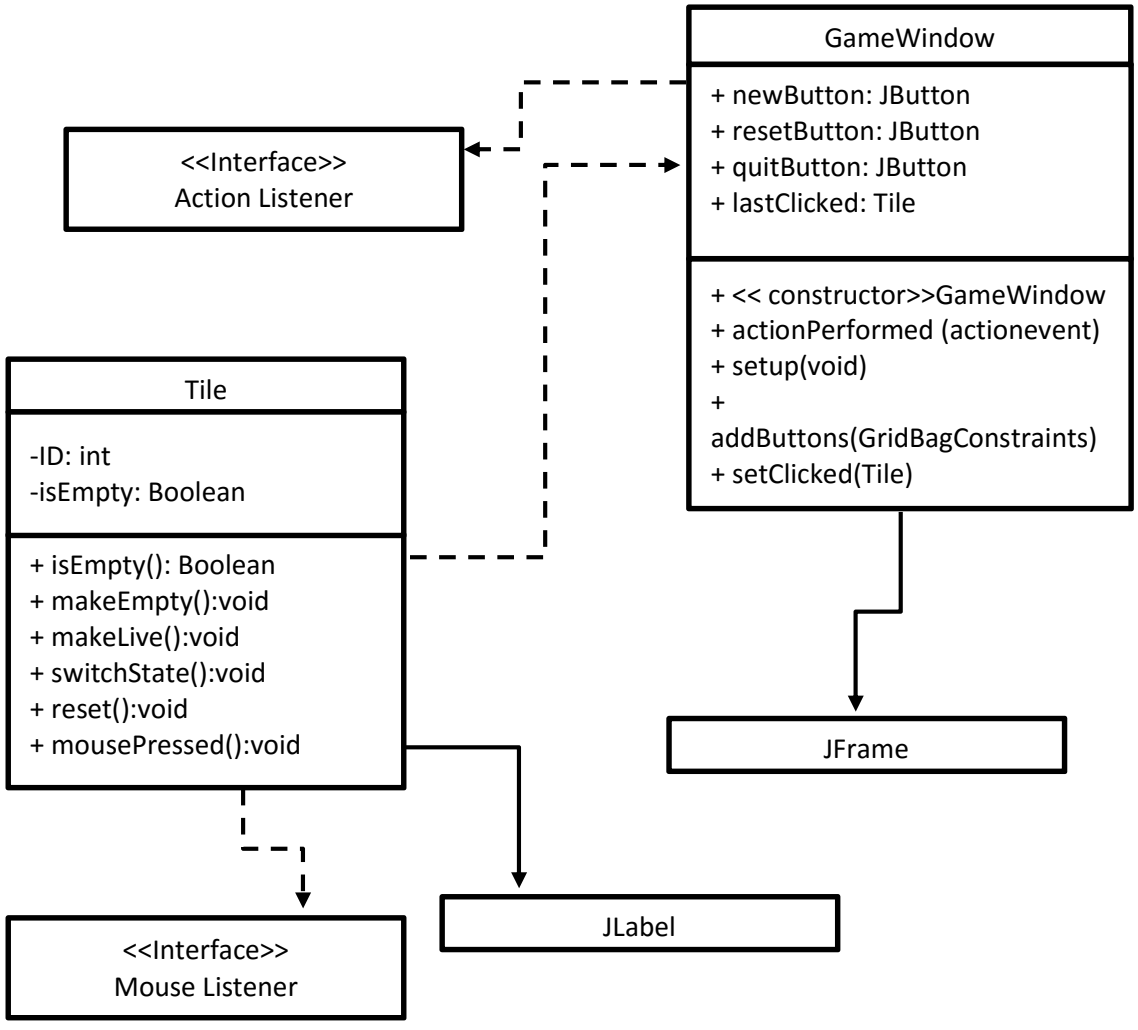
**Program 2 | Interaction**

Next, we made the tiles interchangeable. A Tile could be moved from the starting slot to a game board slot, from a game board slot back to the starting slot, to other slots on the game board, and from a starting slot to another starting slot. This hasn't changed.

Our tile colors were (and still are) set up as follows:

- Empty Tiles ------> grey
- Occupied Tiles --> white
- Selected Tiles ---> green

Occupied Tiles also had an integer ID on them. This is now toggled with "Verbose." Selected Tiles were originally to be selected on –click and was changed to on-select to increase the reliability of clicking on a Tile. Tiles on the game board no longer had borders on them so that they could be matched with adjacent Tiles. The code was altered to remove redundancies and to make it more readable.

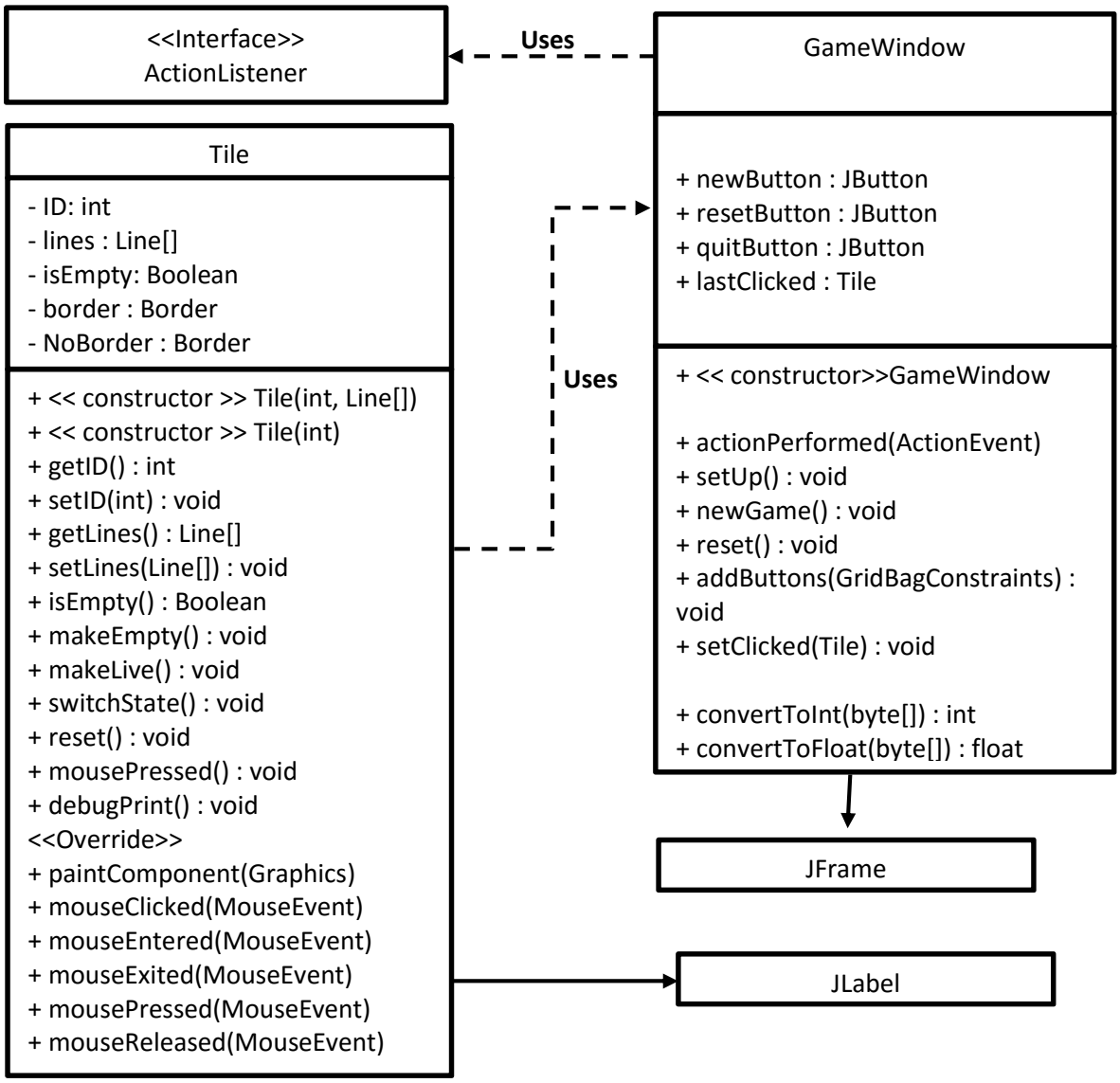


**Program 3 | Input from File**

The first thing we did for this part of the program was read in the raw maze information file. Because the values in the file were ints/floats, we were able to use the convertToInt() and convertToFloat() methods to correctly read the bytes without typecasting. We read in this file successfully, but our code was ugly. Despite constant efforts to clean and maintain the code there were still design flaws.

Next we drew lines on the tiles. These lines matched with the data inside the file and were drawn correctly. We checked with another group as well to make sure that our layout matched the default maze. Assuming they didn't change their code, it should still match.

Lastly, we added functionality to the reset button. This was done in such a way that the tiles all return to their original positions on the sides of the game board. It is not an undo button.



**Program 4 | Randomization**

The next thing we did 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 would then 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. Further parameters where added in later iterations of the project.

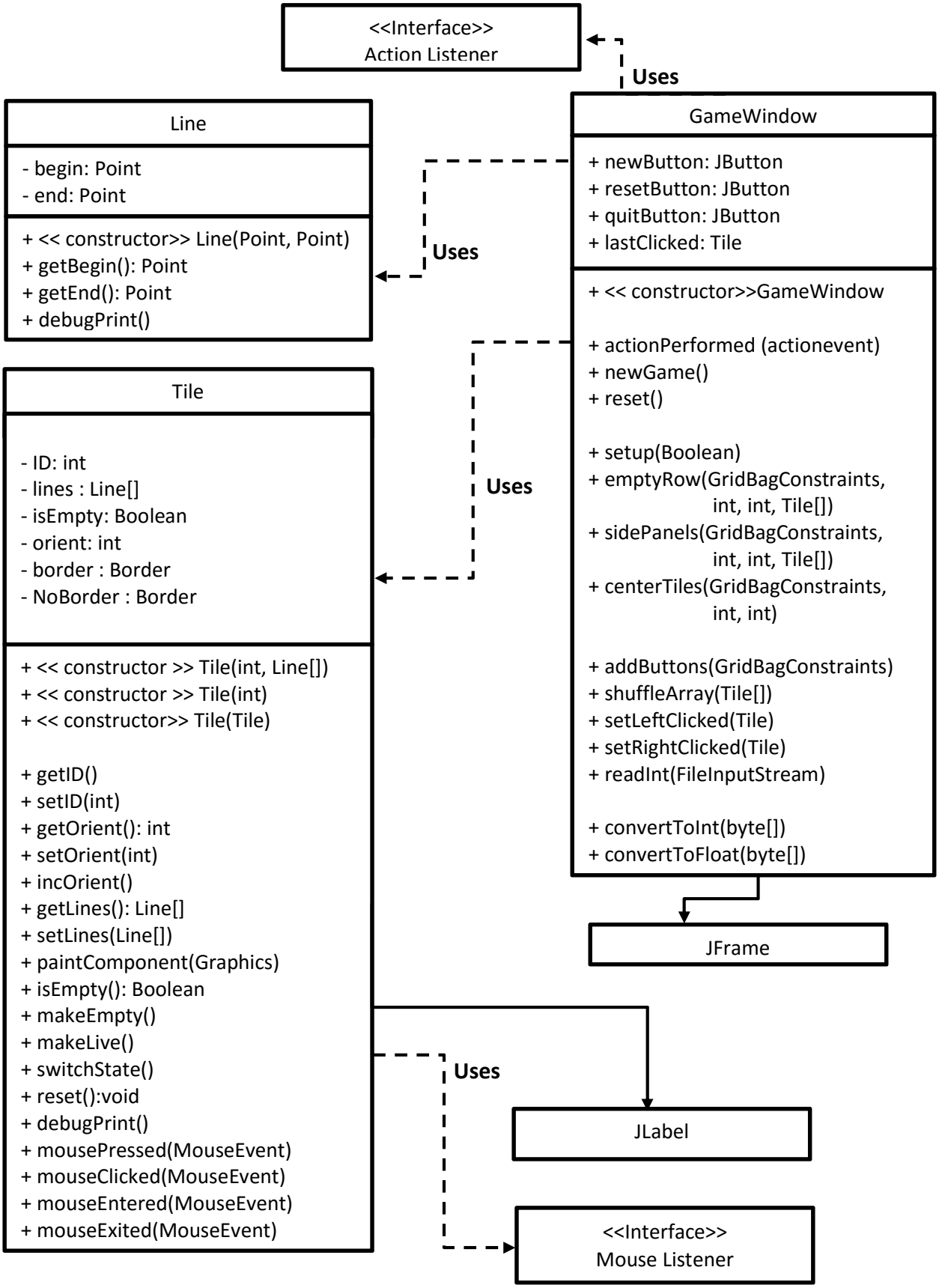
We used this same concept to randomize the orientation of the tiles. Since a tile's orientation never needs to be randomized without the location being randomized and the location being randomized never needs to happen without randomizing the orientation, we were able to use the same parameter for both.

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 (shuffled location and orientation). 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.

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.

(UML on next page)


<<Interface>>
Action Listener
Uses
Line
- begin: Point
- end: Point
+ << constructor>> Line(Point, Point)
+ getBegin(): Point
+ getEnd(): Point
+ debugPrint()
Uses
GameWindow
+ newButton: JButton
+ resetButton: JButton
+ quitButton: JButton
+ lastClicked: Tile
+ << 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[])
Uses
Tile
- ID: int
- lines : Line[]
- isEmpty: Boolean
- orient: int
- border : Border
- NoBorder : Border
+ << 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)
JFrame
Uses
JLabel
<<Interface>>
Mouse Listener

## Program 5 | Load and Save

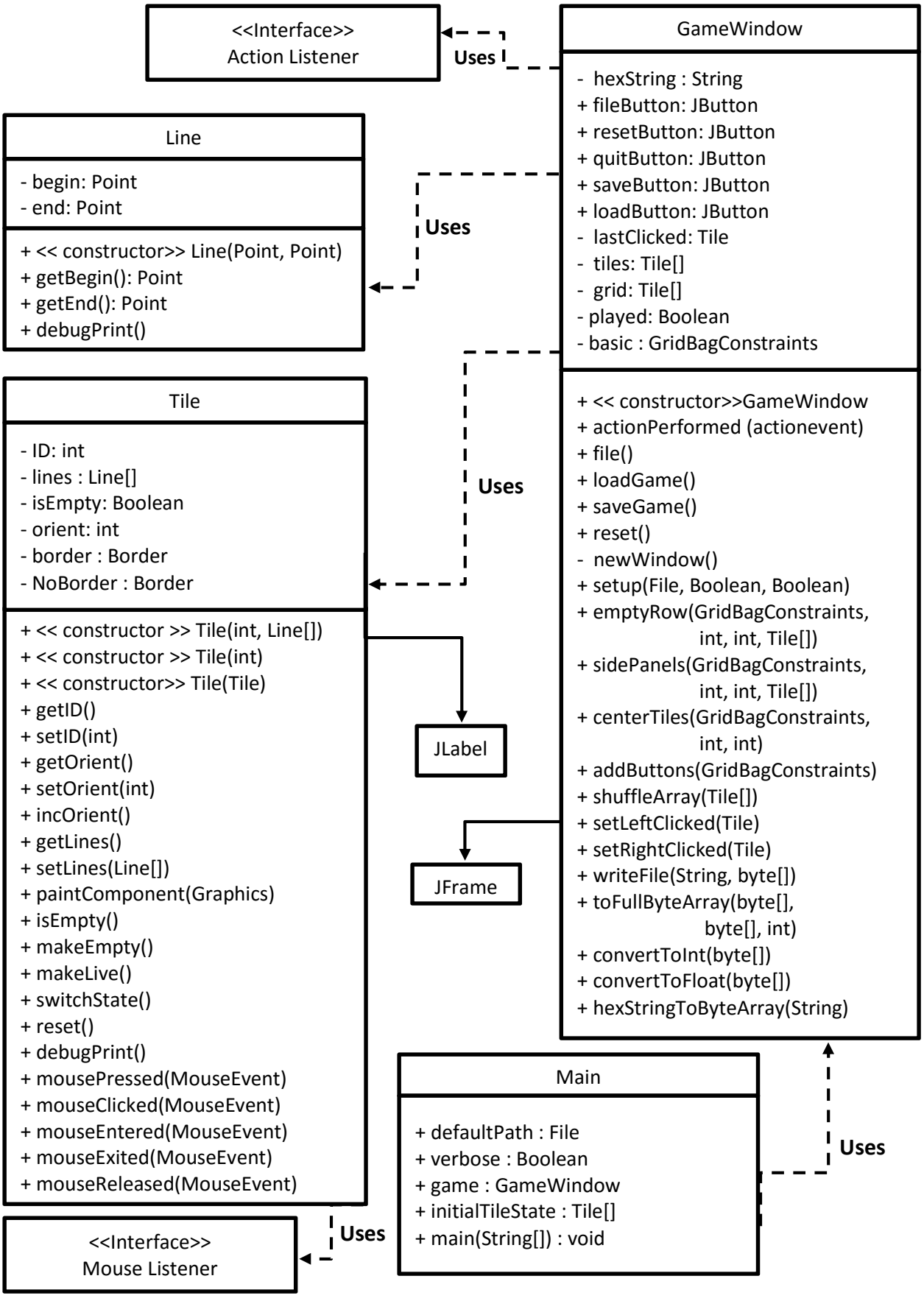
First, we fixed the button layout to disclude “New Game” and include “File,” which leads to two new buttons, “Load” and “Save.” Then, we added functionality to the “Load” and “Save” buttons. The Load button loads a previously saved game, which of course means that the save button saves the current game.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

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.

(UML on next page)


<<Interface>>
Action Listener
Uses
GameWindow
- hexString : String
+ fileButton: JButton
+ resetButton: JButton
+ quitButton: JButton
+ saveButton: JButton
+ loadButton: JButton
- lastClicked: Tile
- tiles: Tile[]
- grid: Tile[]
- played: Boolean
- basic : GridBagConstraints
+ << constructor>>GameWindow
+ actionPerformed (actionevent)
+ file()
+ loadGame()
+ saveGame()
+ reset()
- newWindow()
+ setup(File, Boolean, Boolean)
+ 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)
Line
- begin: Point
- end: Point
+ << constructor>> Line(Point, Point)
+ getBegin(): Point
+ getEnd(): Point
+ debugPrint()
Uses
Tile
- ID: int
- lines : Line[]
- isEmpty: Boolean
- orient: int
- border : Border
- NoBorder : Border
+ << constructor >> Tile(int, Line[])
+ << constructor >> Tile(int)
+ << constructor>> Tile(Tile)
+ 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)
Uses
JLabel
JFrame
Main
+ defaultPath : File
+ verbose : Boolean
+ game : GameWindow
+ initialTileState : Tile[]
+ main(String[]) : void
Uses
<<Interface>>
Mouse Listener
Uses

## Program 6 | New Requirements

- The program implements all of the functionality of the first 5 assignments.
- The game times the player from the first move to winning the game.
- The file format is changed to support time.
- Any file that is being loaded that does not have the correct beginning format triggers an error message.
- The lines corresponding to the read data are printed on the tiles.
- The file reader deals with files based on their first four bytes.
- Each tiles position and rotation is dynamically determined.
- If a user wins the game, a message tells them that they have won and the time it took to solve the puzzle.

## Program 6 | Changes

The first thing we did on the final iteration of the project was fix bugs from program 5. These bugs included:

- The program shouldn't explode if a maze is loaded with a wrong magic number. It should have an error message instead.
- Our buttons should be named "Load" and "Save," and not "Load Game" and "Save Game."
- Ideally, we would prefer to default into the current directory. This is much easier to achieve in Windows than Linux.

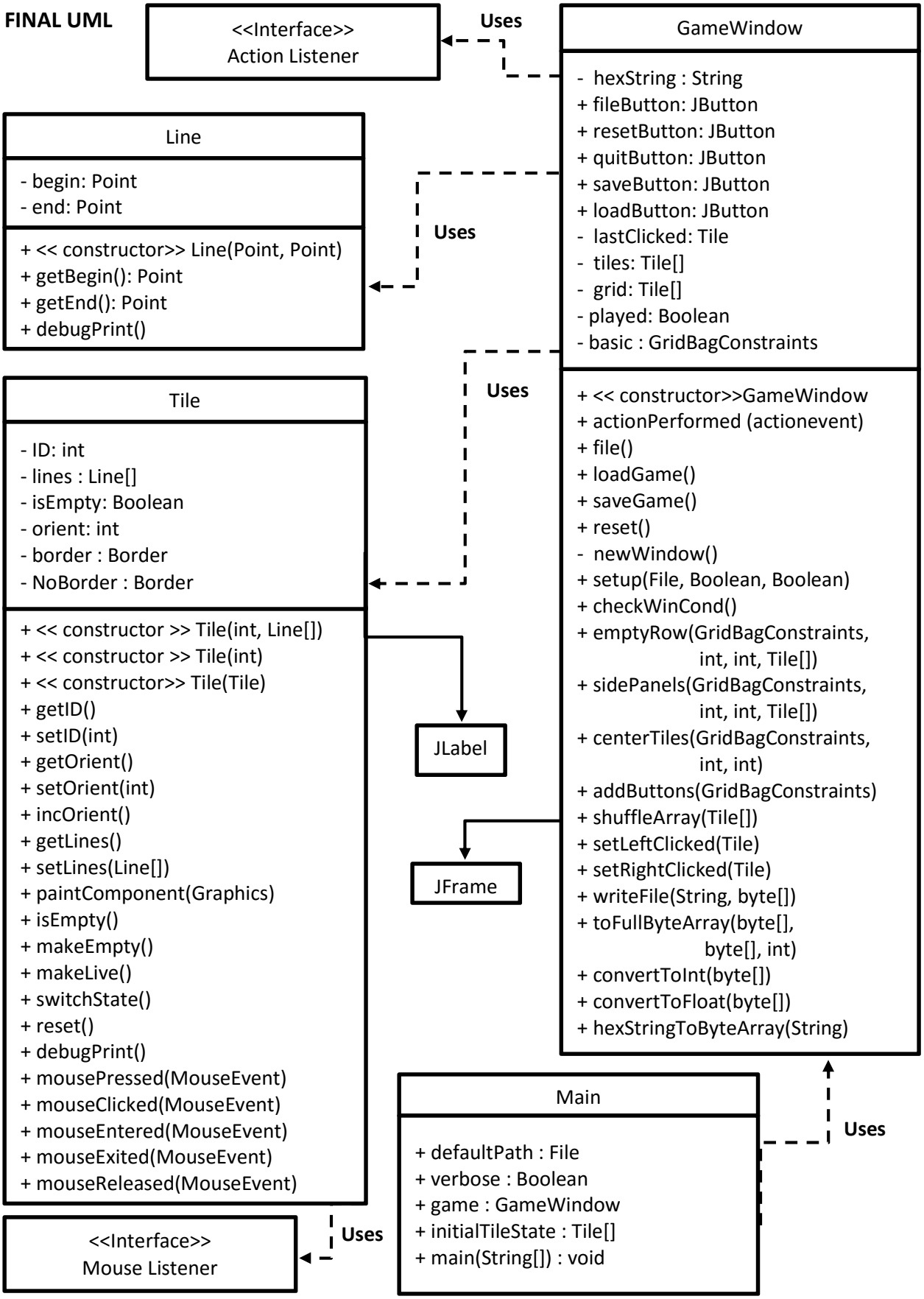
The next thing we did was test our game against all of our previous requirements to ensure that all of the functionality from the first 5 programs was working.

After making sure our program was up-to-date, we implemented the win conditions. These simply check to make sure all of the tiles are in order and that their orientation is 0. This technically doesn't account for the other three cases where the whole board can be rotated but this functionality wasn't as important as others.

We then implemented a timer. This counts up (unlike the timer class in Java) and keeps track of the time from the first move to the move that finally satisfies the win conditions. When the win conditions are satisfied, a pop-up tells the player that they won and how long it took them to complete the puzzle. We made sure that the game does not ask to save if you win the game.

We tested our game extensively on Windows and Linux. The testing took longer than expected but was of course, absolutely necessary.

**FINAL UML**


<<Interface>>
Action Listener
Uses
GameWindow
- hexString : String
+ fileButton: JButton
+ resetButton: JButton
+ quitButton: JButton
+ saveButton: JButton
+ loadButton: JButton
- lastClicked: Tile
- tiles: Tile[]
- grid: Tile[]
- played: Boolean
- basic : GridBagConstraints
+ << constructor>>GameWindow
+ actionPerformed (actionevent)
+ file()
+ loadGame()
+ saveGame()
+ reset()
- newWindow()
+ setup(File, Boolean, Boolean)
+ checkWinCond()
+ 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)
Line
- begin: Point
- end: Point
+ << constructor>> Line(Point, Point)
+ getBegin(): Point
+ getEnd(): Point
+ debugPrint()
Uses
Tile
- ID: int
- lines : Line[]
- isEmpty: Boolean
- orient: int
- border : Border
- NoBorder : Border
+ << constructor >> Tile(int, Line[])
+ << constructor >> Tile(int)
+ << constructor>> Tile(Tile)
+ 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)
Uses
JLabel
JFrame
<<Interface>>
Mouse Listener
Uses
Main
+ defaultPath : File
+ verbose : Boolean
+ game : GameWindow
+ initialTileState : Tile[]
+ main(String[]) : void
Uses