Planning for Group Juliet:

Maze Program Version: 4.0

**Problem:**

For the **first version**, we need to add buttons to the UI, create the gameboard, and add the sidebars. We also need to implement functionality for the quit button. We decided that Adeline, Austin, Michael, and Ben need to create individual and separate branches to experiment with different types of implementations. Ryan will be creating the UML diagram to help organize the layout of the program.

For the **second version**, we need to make the tiles movable and correct the marks from our previous submission. The corrections included resizing the game board and centering it.

For the **third version**, we need to read in the file, draw the maze to its specifications, make the reset button work, and rotate tiles.

For the **fourth version**, we need to add random tile rotation, random tile location in holding areas, make the reset button work with rotation, make the tiles flash to indicate an invalid move, and make several small adjustments to correct for marks on previous assignments.

For the **fifth version,** we need to change the file format, write a played maze to a file in the correct binary format, load a played maze, generate error popups to direct the user through all potential use cases, update the set of buttons to load/save file explorer and adjust all exiting behavior to confirm saves.

**Initial approach:**

Our basic idea is that we need to implement the GridBag class to create the gameboard. We do not need to implement moving pieces yet. However, this will be something we need to work on as well as the ability to resize/scale the game board. We will also have to look into implementing possible exception handling in the future in the event we run into some kind of runtime exception.

**Program Constraints:**

Board: Should be able to store tiles in panels which facilitates the gameplay.

Tiles: Must be visible, move, rotate when right clicked, not swap locations when dropped on an existing tile and contain the drawings of the graph.(Updated 4/11)

Game: This must have a system for determining a valid solution and storing tiles. Michael wasn’t sure as to why it was recommended that we use GridBag. Everyone has a different implementation of GridBag so we will test each implementation to see which one works best. (Update 2/11) We feel good about moving forward with GridBag. The game must offer interaction with the user by allowing tiles to have their location changed and rotated.(Updated 4/11) The user must be able to save their version of the game and return to it. This should have different behavior from if they read in a new game because this cannot be randomized(Updated 4/30).

Buttons: We need to implement functionality for the quit button however, the other buttons just need to be visible. We will come back to implementing functionality to the other buttons at a later date. The reset button now puts the tiles back into the side panel(Update 3/19)Reset button now returns tiles to holding positions with correct rotation (Updated 4/11).There must be a load/save button, reset button and quit button. These are accompanied by a set of logic to warn the user about their decisions like overwriting file, exiting without saving etc.(Updated 4/30).

Sidebars: The sidebars just need to be visible but do not need functionality yet. Adeline sent a screenshot with visible sidebars to show successful implementation. These need to store tile objects and have the ability for those tile objects to be moved. These also need to be numbered(Updated 2/25). The side panels now need to hold tiles(Updated 3/19).

Maze Files: The program needs to work with the specified maze file format(Updated 4/30).

**Solution by Unit:**

Board: Built out of Jpanel objects that get tile objects when they are given to it(Updated 3/19).

Tiles: We need to figure out what kind of object we want to use for the tiles such as JPanels(Update 2/11) We will move forward using JPanels. (Update 2/25) We will use a tile class to isolate their performance from the rest of the board. (Update 3/19) We are using the tile class to manage drawing the tiles on the board. It is important tiles have up to date information about their contents because this information is leverage for writing(Updated 4/30)

Game: Michael wasn’t sure as to why it was recommended that we use GridBag. Everyone has a different implementation of GridBag so we will test each implementation to see which one works best. (Update 2/11) We feel good about moving forward with GridBag. The game now has full functionality where the user can move/rotate tiles. This is still achieved with GridBag(Updated 4/30)

Buttons: Implemented with gridbag button objects(2/11).Button functionality has evolved but still implemented with gridbag(4/30).

Sidebars: Panel objects store them to manage spacing. The holding locations are also built out of panel objects.(Update 3/19)

Maze files: We use the methods given in version 3 to convert between binary and float/int. These are isolated to their own class to avoid code duplication when they are leveraged by the reader and writer. (Updated 4/30).

File management: Files default to the input directory but the user is enabled to load/write files anywhere on their file system.(Updated 4/30)

Overall layout: Managed with a collection of objects from gridbag. Each subunit, gameboard, side panels and nav bar each are in their own panel to allow greater control.(Updated 4/11)

**Deliverable:**

The game is created in Main.java through calls to 11 other classes where the actual organization of all the parts is delegated. We divided the logic of the game into 11 classes, titled BackgroundClickListener, Converter, GameWindow, GameWindowListener, InvalidMazeFileException, PanelClickListener, Nav\_Bar, Converter, MZEReader, MZEWritter, Tile, and TileFlasher. GameWindow is responsible for managing the creation of Tile and Panel objects that make up the entire game. PanelClickListener is responsible for assigning click interactions to different objects (in this iteration just tiles). BackgroundClickListener is responsible for handling when the user clicks on a tile. GameWindowListener is responsible for managing the opening and closing of windows. InvalidMazeFileException is responsible for managing the invalid maze exception. Nav\_Bar handels the buttons bar at the top of the game. Tile contains all of the properties of the game tiles. Tile extends Jpanel because it needs all of the functionality of a panel with added features. Tile and PanelClickListener were designed with extendability in mind and will likely be the focus of modification for future iterations. MZEReader manages the input file. MZEWritter is responsible for collecting all the important information about the game and writing it to a binary file. Converter manages the conversion between the binary data and the array of values given to the Tile class used for drawing. TileFlasher manages the changes that happen after a tile is selected. These parts make the deliverable of the game.

**Future Versions:**

We know there will be significant functionality to add in future versions and the design we chose for this assignment enables this future extendability. Below are future features and how we plan to incorporate them into our existing design.

Game Piece Functionality: The tile storage locations and game board are made using the JPanel class. This choice enables the locations to store objects inside them which is how we plan to manage the game pieces themselves. Furthermore, the Tile class created enables us to build functionality into the Tile objects themselves. We think all functionality has been built into the game tiles but are prepared in-case there is a specification change or another feature.

Button Functionality: The last button that needs functionality is the new game button. This will depend exactly what is required of this button.

Resizability: Our current version does not resize the elements to fit when the window is resized. This problem will likely involve the weightx and weighty options for gridbag and we think our current organization lends itself toward that.

Win Conditions: We know there will have to be a way of determining if a user has won the game. This will involve recording the rotation of the tiles and their location. Our tile objects are equipped to store information about their orientation and location. Then all we will need to add is a method that checks this against an answer key.

=> Other components could be added that would affect win conditions of the game like different sized grids or the maze must be completed in a set period of time.

Unforeseen Additions: We know there will be things to add that we are unaware of right now. We have done 2 things to account for these changes. First, we built our entire game with extendability in mind by creating units that can have new features added that do not impact the other parts of the game. Second, we chose to put certain things like the tiles in their own class so those objects can be manipulated specifically without needing to change other features.

Features we a want to see:

--Different difficulty levels based around a timer that could count down or up in different game modes.

--Different difficulty levels that change the size of the grid.

--A difficulty level that tells you when a single tile is placed in the correct location to make the game easier.

-- Different styles of lines, like smooth curves to change the type of maze you are building