Planning for Group Juliet:

Final Maze Program

**Problem:**

*A high level view of what was addressed in each iteration of the project*

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.

For the **sixth version**, the most important addition is winning the game and a timing system. This involves modifying how files are read, saved, move tracking is conducted. This is the last version so the game should be in a usable state by the end of this iteration.

**Initial approach:**

*This section only outlines our initial approach to the program (not the final product)*

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:**

*This section includes how the constraints have evolved over time into the final product.*

Board: Should be able to store tiles in panels which facilitates the gameplay(Updated 4/11).

Tiles: Must be visible, move, rotate when right clicked, not swap locations when dropped on an existing tile and contain the drawings of the maze. The tiles must also visually indicate to users when an invalid move has been attempted(Updated 4/11). Tiles should have some awareness of their location in the board for knowing when the maze is complete(Updated 5/7).

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). The game must track time and notify the user when it is complete. The game also must store all of the features about the play state(time, location, rotation) to the file when the file is saved.(Updated 5/7)

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 a 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). If the side bars contain tiles the game cannot be complete(Updated 5/7)

Maze Files: The program needs to work with the specified maze file format(Updated 4/30). The maze files must now be written in the answer key order and store the time of the game when saved(Updated 5/7).

Time: Time is shown to the player at the top of the screen and counts up once the maze game has been started. It should be retained when a maze is saved and reloaded. A loaded maze should only have its time reset to the value in the saved file. The time should be stopped when the user is loading or saving file.(Updated 5/7)

**Solution by Unit:**

*Outlines how each unit of the game was addressed without referencing the source code*

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) This information now includes a home position to know when the game is finished(Updated 5/7).

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). The user is notified when they have won the game by checking the tiles location against the answer key after each move(Updated 5/13).

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). Maze files now write in the correct order and retain the time elapsed. (Updated 5/7)

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)

Timer: This is managed with a class that extends the TimerTask class and calls in Game Window (Updated 5/13)

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 Delegation of Work:**

*Outlines how the game is broken into classes that address different functionality. Could be used as a high level guide to the actual source code.*

The game is created in Main.java through calls to 12 other classes where the actual organization of all the parts is delegated. We divided the logic of the game into the 12 classes, titled BackgroundClickListener, Converter, GameTimerTask, GameWindow, GameWindowListener, InvalidMazeFileException, MZEReader, MZEWritter, Nav\_Bar, PanelClickListener, Tile, and TileFlasher.

GameWindow is responsible for managing the creation of Tile and Panel objects that make up the entire game. It is also responsible for maintaining information about the state of the game. This includes the time, tile objects, and if the game has been won. pannelClickListener is responsible for assigning click interactions to tiles. BackgroundClickListener is responsible for handling when the user clicks on a tile. GameWindowListener is responsible for managing the opening and closing of windows. GameTimeTask manages the timer elements. 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. Creating a separate converter class allowed us to localize that information to one place instead of duplicating it in reader and writer. TileFlasher manages the changes that happen after a tile is selected. These parts make the deliverable of the game.

**Future:**

*How the game was built to enable extendability and what features we’d still like to see added.*

While the game is now finished it was still built with future extendability in mind. Below are how we built elements for extendability.

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: New buttons can be added to the gridbag layout and existing buttons can be renamed with a single line of code.

Maze files. The writing process is decently flexible but requires modifications to existing methods if someone wants to record new information. This was a tradeoff between increasing complexity of the existing solution in expectation of change and we chose to maintain the simplicity of the current solution.

Unforeseen Additions: 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. Third, we keep data manipulation mostly within its own class to help reduce cascades of errors that are a product of one change. Over the 6 design iterations this approach facilitated changing to new requirements (ex modifying the the maze files)

Features we want to see someday:

--Different difficulty levels like 3x3 grid or 5x5 grid.

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

-- A leaderboard that tells the high score for different difficulty levels

-- An algorithm that will generate new .mze files so the game can be played an arbitrary number of times with completely new tiles.

-- A way to play with friends where you both try to solve the same maze at the same time.

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