Version 1

Group: Foxtrot

Maze Program Version: 1.0

Our basic idea for this version is to create the gameboard, the piece spots, and the buttons and to have them all in the correct positions.

Tiles: We are not ready to implement movable tiles, so this will be saved for future versions.

Gameboard: Gridbag layout was difficult to learn, but Buck figured out how to set the pieces in the correct position. Getting these set up and working was simple once we understood the GridBagConstraints function, especially the tile pieces. We also changed the background color to a less vibrant color.

Buttons: Getting the buttons to work was simple enough, as we coded the public class for the buttons, since they will be used throughout the project. The exit button works as intended, but the start and reset button still do nothing other than output to the console. These other buttons will be addressed in future versions.

Sidebars: These were created alongside the Game Board. They are just JPanles right now but will eventually become more and hold the pieces that we need.

Entire Project

Planning for Group Foxtrot:

Create a maze game that has tiles that need to be arranged in a path to win.

Win Condition: To create a win condition for the maze game, a checker needs to be implemented that verifies if the tiles are arranged the correct way for all possible win conditions. This will allow players to know when they have successfully completed the game.

Window Size: Setting a minimum or static window size for the game is important to provide a consistent display experience for players, so that even when resizing it will make sense to the user. This will help ensure that the game can be played on various screen sizes, and accommodate players screen size preferences..

Moving Tiles: The tiles in the game need to be designed to be movable and rotatable within the game board. We will achieve this by creating code that allows the tiles to change positions. We will have to keep track of all current positions of tiles and their rotations, which will eventually be connected with a win checker to determine if everything is in its correct position.

Tile Randomizer: A tile randomizer should be added to the game to create different paths each time it is played. We will do this by having a class that randomizes the rotation of each individual tile piece and which position in the “sidebar” it will be located. It should make sure that every new game looks different to the user.

Win Screen: The win screen should be designed to display when the player has successfully arranged the tiles in the correct path. This will provide a visual confirmation of the player's success and create a sense of accomplishment for them. It will allow them to exit the game or start a new game if they please.

UML Diagram: The UML should be reviewed and be updated as necessary to ensure it accurately reflects the design and implementation of the game for each step of the way.

Bug Fixes: Regular bug fixes should be performed to maintain the stability and functionality of the game. This will ensure that players have a smooth experience while playing the game and that the code looks nice, readable, and maintains Object Oriented principles.

Version 2

Group: Foxtrot

Maze Program Version: 2.0

The idea for this version was to add tile pieces that could be moved from the side bars to the board.
We also fixed issues brought up from the previous programming assignment.

Tile:

Added game tiles. Currently these are Jpanels that use a mouse listener for when they are clicked.

These display a number and color when implemented.

The tiles change colors when selected to allow the user to know which one is clicked.

Tile Functionality:

Use one click on the left button on the mouse to select the tile, and use one click on the a playbox to place the tile in its slot.

Playbox:

This is a layered JPanel that creates the starting points for the tiles. This gives us a great deal of flexibility since we can create borders on one layer and cover them with a slightly larger tile.

Added grid lines to show where the tiles can be placed.

Board:

This is also a set of layered JPanels that provides a play area for the tiles.

Added grid lines to show where tiles can be played.

Buttons:

Fixed the “New” button so it now displays “New Game” and calls for an action of the same name.

Removed the Println commands.

All Documents:

Removed tabs and checked for line length.

Edited documentation

UML Diagram:

Fixed the way interfaces were displayed.

Added new classes and methods to the diagram.

Rest of the Project:

Tile:

Create a rotation method. We will likely have to draw something on the Jpanel and move the components

according to the rotation value.

We may want to change the selection to an outline or a slightly opaque cover in the future based on what the tile design ends up being.

Win Conditions:

Still need to determine what win conditions will be and how they will be checked. This will deal with both tile placement and rotation. So a method will have to check whenever the board is full and another whether the tiles are in a position to win or not.The win rotations and placements could be stored in a data structure or a text document. We will have to discuss this more in-depth later.

We should also add a pop up or animation that lets the user know they've won and asks them if they Would like to start a new game, restart, or quit.

Buttons:

Need to make sure New Game and Reset work. There will likely be methods to clear the board and fill each tile holder with a random tile. Something needs to hold the default positions and rotations for tiles when a game starts. This may be as simple as keeping the values in an array and moving the pieces back to the correct positions when reset is pressed.

Tile Randomizer:

This will be used when New Game is pressed. This will assign each tile a random spot when the game starts.

Game Window:

We still need to look into making the window scaleable or have a size adjustment option. This will have to be checked on multiple monitors.

Version 3

Group: Foxtrot

Maze Program Version: 3.0

The main idea for this version was to replace the generic tiles with maze pieces and creating a working reset button.

For the maze pieces we had to read data from a binary file and draw lines on our game tiles according to the data in the file. After figuring out the format of the data this was a fairly simple task. All we did was pass this data as a parameter to the Tile class and it utilized a draw function to draw the lines on the tile.

For the reset button we simply stored the tiles in an array and added them back to the starting panels when the reset button is pressed. We also cleared the play area.

Another thing that changed in this version is how borders are handled. As there are adjacent borders from adjacent playboxes we need to be able to turn these off correctly when a tile is placed in a playbox ie. eight borders need to be turned off instead of just four. This is implemented with a removeBorders function in the playbox class which removes the eight (possibly less for edge tiles) borders. For adding the correct borders back if a tile is moved from a grid playbox the program draws borders around every playbox and removeBorders is then called on all the playboxes with tiles in them. This accurately turns on and off borders depending on where a tile is.

Outside of functionality changes, there were also changes to the documentation of the code. The new styling works with doxygen.

Rest of the Project:

Tile:

We still need to make a rotation method for the tiles that will move the lines drawn on the tiles to a new location as they are rotated.

Win Condition:

Our plans are still the same as the last two versions. There is nothing more to extrapolate here.

Buttons:

New Game is the only other button that we need to make work. This button will give tiles a random starting position and rotation value.

The reset button's functionality will have to be expanded upon when we add rotation to the pieces. Currently, it is only concerned about each tile's starting position.

Game Window:

With a gridbag layout the components scale based on the display. Making the components a reasonable size should be the main concern.

Version 4

Group: Foxtrot

Maze Program Version: 4.0

In this update, our primary objective was to enhance the gameplay experience by incorporating randomized tile positions and rotations in the holding areas, delivering a unique and engaging challenge for players. Furthermore, we introduced the TileFlasher class to provide interactive feedback on specific events and undertook a thorough refactoring of the existing code to better adhere to OOP principles, ensuring a more organized and maintainable code.

Randomizer: Our randomization now shuffles the positions and rotations of the tiles in the holding areas, offering a fresh and unpredictable maze for players to solve each time they load the game. This feature is achieved through the use of an array and a list that are read in and shuffled with java's built in functions.

Tile: We made necessary modifications to the Tile class to accommodate the new functionality introduced by the Randomizer class, primarily focusing on each tile's rotation.

TileFlasher: This class listens for event actions and causes tiles to flash on a timer with a delay, adding an extra layer of interaction for the players so they know when they tried to make an invalid move.

FileSetup: We restructured this as a standalone class for improved organization.

PlayAreas: Resets to a normal state using the method resetTiles, which takes indices and rotation arguments as specified in Randomizer in a list and array respectively.

All Documents: We further enhanced the Javadoc comments for better annotation of new elements.

UML: We updated the diagram to include the new methods and classes that were implemented this time around..

Rest of the Project:

Buttons: We discussed the possibility of moving the 'New Game' and 'Restart' buttons into separate classes to streamline the Game Window and focus more on display-related functionality.

TileFlasher: We considered merging this class into the Tile class for better organization.

Win Checker: In future updates, we plan to create a mechanism that checks for win conditions and provides a notification upon winning. It will need to check the 4 possible win orientations of the maze.

Bug/Error Handling: As we move forward, we will work on creating additional safety nets for expected errors and address any bugs that may arise.

Version 5

Group: Foxtrot

Maze Program Version: 5.0

In this version we strove to implement saving and loading of tiles' properties to external documents, as well as providing a window that would help the player save and load various different maze configurations from these files by supplying the program with a valid filename. Additionally we changed our reset button to reset the game to its original loaded position in the file instead of a default configuration.

"New Game" Button: Changed the button to display "File", which brings up the load menu when clicked.

Load: This method loads a maze configuration including placement and rotation (depending of the magic number) of the tile objects within the play area.

Save: Saves the current maze configuration to a file following the new guidelines and displays the same file menu for the user to work with.

Refactoring: Some functions were made into their own classes to help with the modularity and readability of the code. This makes the coding of the process easier, but the "connections" between the classes could use some work.

Game Functionality: Different combinations of actions were implemented into the game. For example if a player loads a "bad" maze file, nothing displays, then the user tries to save the game when there is nothing, the game will display an error message. All these different player actions were accounted for so that the user does not run into any errors when playing the game in its current state.

UML Diagram: Updated the UML diagram with the inclusion of the new methods that we added for Saving and Loading.

Rest of the Project:

Win Condition: We will look to add a win condition to the assignment in the future.

Refactoring: The code is a bit convoluted, with multiple references across classes that could stand to be more streamline.

Bugs/Errors: Handling of errors could still be further expanded upon, and with the inclusion of more user input, there is a greater chance of unforeseen bugs popping up. We hope to see these in further development and catch them