Documentation

Displaying the data:

* The initial grid size is passed through the program’s main method. This argument is used to initialize the class’s marupeke grid object with a random marupeke grid of the specified size.
  + The randomPuzzle() method was limited to only generate finishable puzzles when the size is less than 6 (3-6). Larger puzzles are only legal as generating finishable grids for them takes a long time.
* The GUI uses a BorderPane as the root of the scene.
* Two different VBox objects (one of which contains an HBox) are used in the GUI. With one set as the top of the BorderPane and the other being set as the right of the BorderPane.
* The center of the BorderPane stores a GridPane which is used for displaying the actual puzzle grid.
* A call to updateGrid() is used to first fill the GridPane with buttons that act as the grid’s tiles.
* This method accesses the grid’s marupekeTiles 2D array field and uses it to initialize each button in the GridPane.
* Each time a call to updateGrid() is made, the GridPane is first cleared before it is again filled with the buttons in their appropriate state.

Editing the data:

* Every editable button in the GridPane is configured with an event handler used for editing the tile’s data.
* When the user thus presses a button, multiple things happen:
  + If the user has previously pressed the unmark button, the button’s text is set to blank (“ ”), the appropriate position in the marupekeGrid object is unmarked, and the unmark conditional used for checking if the unmark button was pressed or not is set to false.
  + Otherwise, the event handler checks whether the button press was a primary (left) or secondary (right) click.
    - If the button press was a primary press, the button’s text is set to a cross (“X”), and the appropriate position in the marupekeGrid object is also marked with an X (through markX() to allow future edits).
    - If the button press was a secondary press, the button’s text is set to a circle (“O”), and the appropriate position in the marupekeGrid object is also marked with an O (through markO() to allow future edits).
  + If the current state of the grid is illegal but the grid is not filled, an alert pops up indicating that.
  + If the grid is filled:
    - If the grid is also complete, a congratulations alert pops up with the option to play again.
    - Otherwise, an alert pops up telling the user that the puzzle was filled illegally, along with the option to play again.
* A single call is made to updateGrid() whenever the generate button is clicked with a size selected through the valid sizes combo box. Clearing the GridPane ensures that no buttons are left from the previous grid.
* A single call is made to updateGrid() whenever the dark or light mode buttons are pressed.

Optional extras:

* Optionally, the user can pass 4 arguments instead of one to the main method in order to create a more specific marupeke grid.
* 2 themes were available for the grid, a dark mode, and a light mode. The buttons themselves are within the VBox on the right side of the BorderPane. However, the implementations of each theme are within the updateGrid() method (which is called every time one of the theme/mode buttons is pressed). When one of the dark mode or light mode buttons is pressed the value for the isDarkMode Boolean field is set to either true or false, respectively. Within the updateGrid() method, the style of each button in GridPane is set to the appropriate style according to whether isDarkMode is true or false. The proper implementation of the updateGrid() method also allows for the symbol of each editable tile to remain after changing the theme. This feature provides nice customizability which also helps with battling eye strain as it allows the user to choose the theme which better suits their current environment.
* Non-editable tiles were darkened in each theme in order to distinguish them from editable tiles, this was also a part of each theme’s implementation.
* Throughout the whole GUI, CSS and Font styling were implemented and used. These styles were implemented through the setStyle(), setTextFill(), and setFont() methods of different GUI elements within the program. This feature allowed for the creation of a more visually pleasing program that is consistent in styling throughout.
* A simple help menu accessed through a button was added to the right VBox. This was implemented by showing an alert with instructions explaining how to win, the distinction between editable and non-editable tiles, and how to use the game’s controls to mark a tile with an X or an O, and how to unmark.
* An elapsed time stopwatch was added at the top of BorderPane’s top VBox. This feature was implemented through the use of labels, text fields, and an animation timer. The 3 text fields were made un-editable and are used to store the current elapsed time while the labels were used to indicate which text field is seconds, minutes, and hours. The animation timer supplies the current elapsed time % 60 to the elapsedSeconds text field. A listener on elapsedSeconds’ textProperty then increments the integer value of elapsedMinutes each time elapsedSeconds reaches 60. Similarly, a listener on elapsedMinutes’s textProperty increments the integer value of elapsedHours each time elapsedMinutes reaches 60. The 2 additional start and stop buttons can be used to start and stop the timer. This feature allows the user to manage and keep track of the amount of time spent playing the game.
* Sound effects were added to the program and are played in correspondence with different user actions. These sound effects were implemented through the AudioClip class, which is well suited for playing short audio, and in this case for a single cycle. This feature enhances the user experience while playing the game as it provides responsive sound effects which indicate that an action was made and that something was done.
  + The changeTile sound effect is played whenever a tile is edited (marked with an X or an O or unmarked)
  + The error sound effect is played whenever a mark causes the grid to be illegal, or when the grid is filled but is not complete (filled illegally).
  + The puzzleComplete sound effect is played when the user solves the puzzle with no illegalities present.
* Each window opened within the program used the same icon throughout. This was implemented through first setting primary stage’s icon. This icon is then used to set the icon of each window (as in alerts or dialogs which pop up). This feature further builds upon the visual consistency of different parts of the GUI.

References:

Icon:

Jimm (n.d.). *Flow XO vector (SVG) logo | Download on logowiki.net*. [online] logowiki.net. Available at: https://logowiki.net/logo/flow-xo [Accessed 30 Apr. 2021].

Error sound effect:

Onaywon (2016). *incorrect sound effect*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=3Aq\_Uck8HTI&ab\_channel=Onaywon [Accessed 30 Apr. 2021].

Change tile sound effect:

Sound Effects SoundCuration (2019). *UI Button Click Sound Effect /Button*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=paHzZwBd3uU&ab\_channel=SoundEffectsSoundCuration [Accessed 30 Apr. 2021].

Puzzle complete sound effect:

Sounds Recorded (2016). *Success 1 Sound Effect*. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=3wr8ntTQaeA&ab\_channel=SoundsRecorded [Accessed 30 Apr. 2021].