Contents

[1. Trouble Shooting 2](#_Toc69990993)

[1.1 Operating system 2](#_Toc69990994)

[1.2 SDK 2](#_Toc69990995)

[1.2.1 Recommended SDK 2](#_Toc69990996)

[1.2.2 Known SDK issues 2](#_Toc69990997)

[1.3 Application 2](#_Toc69990998)

[1.3.1 Grid Generation Timeout 2](#_Toc69990999)

[1.3.2 Illegal Parameters 2](#_Toc69991000)

[2. Command Line Parameters 2](#_Toc69991001)

[2.1 Four Parameters 2](#_Toc69991002)

[2.2 One Parameter 2](#_Toc69991003)

[2.3 No Parameters 2](#_Toc69991004)

[3. Displaying the Data 3](#_Toc69991005)

[3.1 How the Data is Displayed by the GUI 3](#_Toc69991006)

[3.2 How the Data is Fetched 3](#_Toc69991007)

[3.3 How the GUI is Kept up to Date 3](#_Toc69991008)

[4. Editing the Data 3](#_Toc69991009)

[4.1 Marking a Tile with X 3](#_Toc69991010)

[4.2 Marking a Tile with O 3](#_Toc69991011)

[4.3 Clearing a Tile 4](#_Toc69991012)

[5. Optional Extras 4](#_Toc69991013)

[5.1 Animations 4](#_Toc69991014)

[5.3 Hints for the User 4](#_Toc69991015)

[5.4 Quit Button 4](#_Toc69991016)

[5.5 Saving and Loading a Grid 4](#_Toc69991017)

[5.6 Music and Sound Effects 5](#_Toc69991018)

[5.7 Loading Popup 5](#_Toc69991019)

[6. Bibliography 6](#_Toc69991020)

# 1. Trouble Shooting

## 1.1 Operating system

This application is confirmed to run on Windows 10 home version 20H2. It has not been tested on any other operating systems but can be assumed to run on any system with the correct Java SDK installed (see section 1.2.1).

## 1.2 SDK

### 1.2.1 Recommended SDK

This application is confirmed to run on the JDK 8 (1.8) from oracle, available at the following address:

<https://www.oracle.com/uk/java/technologies/javase/javase-jdk8-downloads.html>

### 1.2.2 Known SDK issues

When tested with another SDK, specifically Corretto-1.8 (One of the SDKs available for download through IntelliJ), the program failed to run due to an UnsatisfiedLinkError, caused by a MediaPlayer object. This is possibly due to this SDK having an incomplete set of JavaFX libraries, but the exact cause of the issue is yet to be determined. As a result, the author of the application recommends assessment/marking be conducted using the recommended SDK (see section 1.2.1).

## 1.3 Application

### 1.3.1 Grid Generation Timeout

With larger grid sizes, it is possible that grid generation will timeout. If this happens, an error message will be displayed informing the user. This is common with larger grid sizes (8, 9, 10) when using custom parameters for numFill, numX and numO. To use grids of these sizes, it is best to launch the program with only a size parameter (see section 2.2).

### 1.3.2 Illegal Parameters

If launched with illegal parameters, the program won’t run. The number of marked tiles in the grid cannot exceed (so for a grid of size five, twenty-five tiles or less must me marked).

# 2. Command Line Parameters

## 2.1 Four Parameters

The application can take 4 command line parameters for size (the size of the Marupeke grid), numFill (the number of filled tiles), numX (the number of uneditable X tiles) and numO (the number of uneditable O tiles).

## 2.2 One Parameter

The program will also work if only a size parameter is given. The other parameters will be calculated from the grid size to equal approximately a third of the legal number of tiles (half the tiles in the grid). Note that if less than four, but more than one parameters are given, all parameters other than the size parameter will be ignored. This parameter must be between 3 and 10 inclusive.

## 2.3 No Parameters

The application can launch with no parameters. In this case size will be a random number in the range of 3 – 10 inclusive. In this case, numFill, numX and numO will be generated from the random size parameter in the same way as they would be with a user provided size parameter, except if size is equal to 10, in which case they will all be set to 7. This is because higher parameters make load time increase beyond the timeout time of the grid generation.

# 3. Displaying the Data

## 3.1 How the Data is Displayed by the GUI

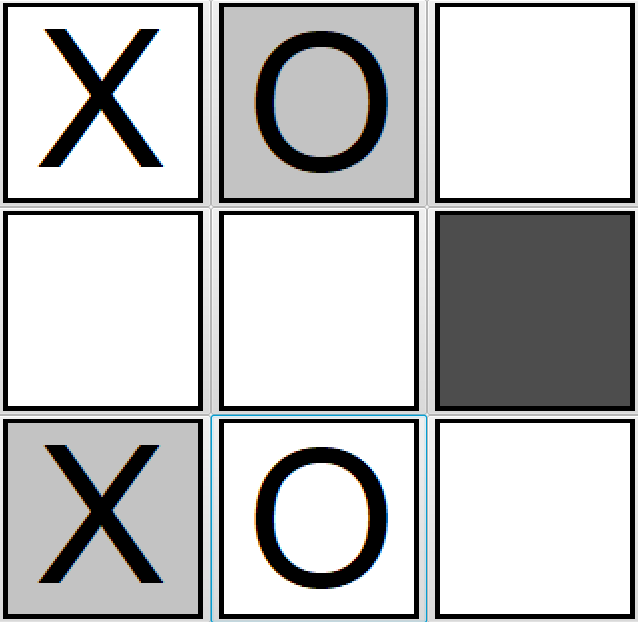
The data is displayed by a GridPane of Button objects. These Button objects each have a different image attached to them depending on the type of data, as shown in figure 1. The dark grey tile is a SOLID tile. The white tiles are BLANK tiles. The O and X on grey tiles are uneditable X and O tiles, and the X and O on white are editable tiles.

Figure The MarupekeGrid as displayed by the GUI

## 3.2 How the Data is Fetched

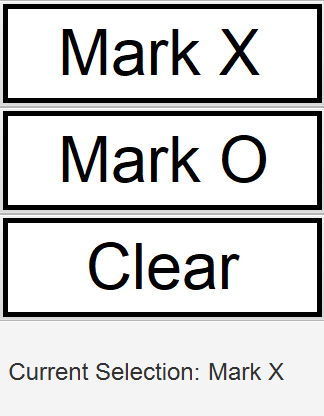
The grid buttons are constructed using an embedded for-loop. The image for button (i, j) is gathered from the grid by a method called getTileImage. This method takes an x and a y parameter and returns the correct image for the tile at location (i, j) in the grid.

## 3.3 How the GUI is Kept up to Date

The GUI is kept up to date by the action event set to the grid tiles. When the event is triggered by a click from the user, the getTileImage method is called for that button to update it. There is also a defaultSetAllTiles method which sets every time in the board to the correct image based on the underlying grid in the model. This is used for resetting the grid to its appropriate state after an animation has played.

# 4. Editing the Data

## 4.1 Marking a Tile with X

When the user opts to mark a tile with an X, they click the Mark X button, as shown in figure one. This button sets the selector field of the controller to X. Then, when they click on a tile, the markTile method of the controller is called. This method marks the tile with the type as dictated by the selector, which in this case is X. Once it has marked the tile, it tests the grid for illegalities. If none are found, then the tile is left how the user set it in the grid. If illegalities are found, they are added to a list, and the tile is unmarked. The list of illegalities is used by the GUI to visually inform the user of the illegalities (see section 3.1).

## 4.2 Marking a Tile with O

Marking a tile with O works in exactly the same way as marking a tile with X, An O is placed in the grid. The user uses the Mark O button to set the selector field to O. This causes the switch statement in markTile to place an O in the grid rather than an X.

Figure Input selection panel from the Marupeke application

## 4.3 Clearing a Tile

Like with marking a tile with an X or an O, clearing a tile sets the selector field of the controller to its respective type, CLEAR. In the switch statement in markTile, the default case of the switch statement simply calls unmark on the tile of the grid.

# 5. Optional Extras

## 5.1 Animations

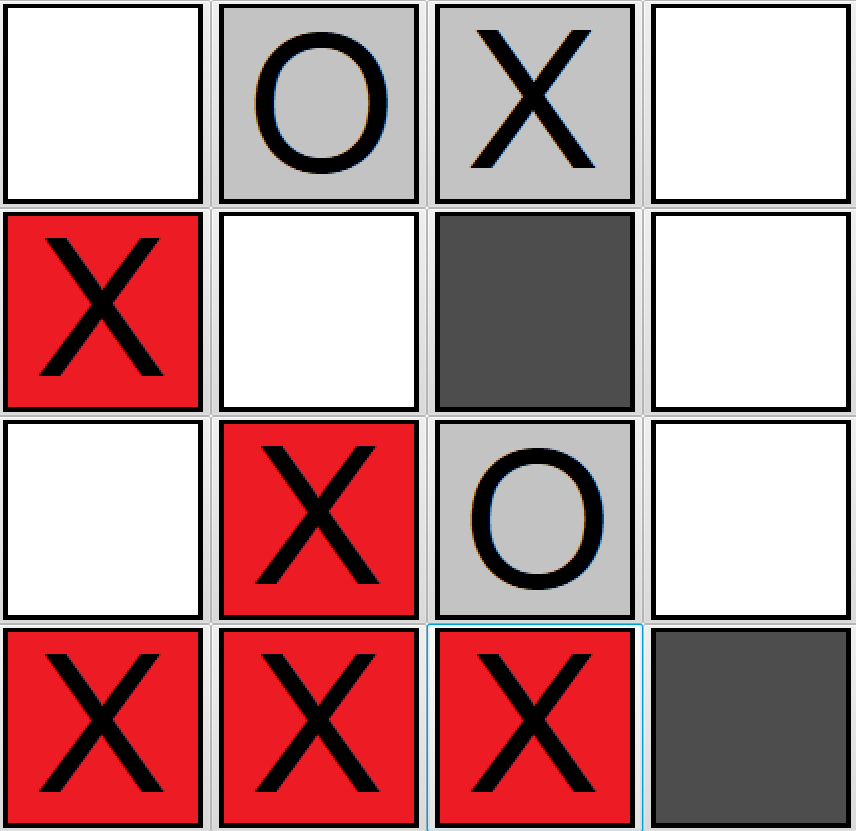
The game features some animation to display information to the user. These animations come in the form of tiles and buttons changing colour. One such animation is the violation animation. The controller contains a list of the violations detected in the grid. A listener is attached to this. When it changes, the program calls the playViolationAnnimation method. This causes all of the tiles in the list of violations to turn red on the screen for several seconds, as shown in figure 3. This accompanied by an audio que (see section 3.6) makes it very clear to the user which tiles are causing the violation. Another animation is used to signify the end of the game. All the editable tiles turn green. This is also accompanied by an audio que. The last animation plays when a hint is displayed (see section 3.3). There are two possible hint animations. If a hint is successfully generated, it appears in the grid as a green tile of the symbol specified by the hint. If no hint can be generated, the hint button (see figure 5) itself changes to a prompt informing the user that they need to unmark some tiles for a hint to be successfully generated.

Figure Two-way violation displayed in the Marupeke grid

## 5.3 Hints for the User

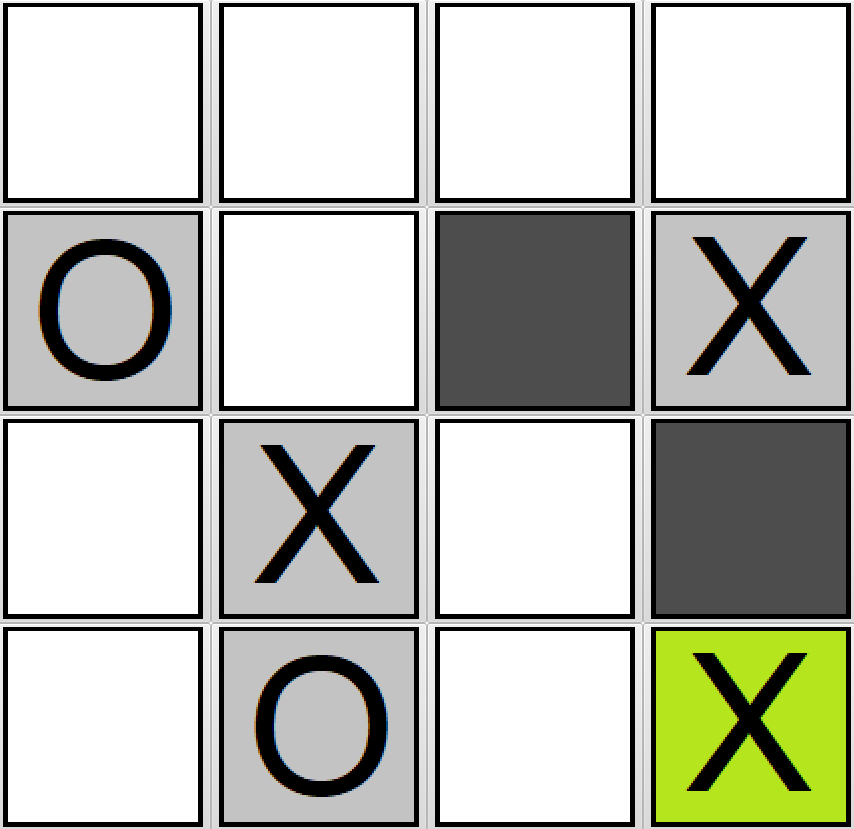
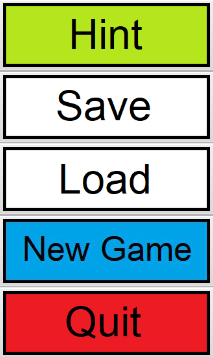
The game implements a hint feature to help players complete the game. the hint is returned by a method called getHint in the controller and stored in a variable for comparison. The hint feature makes use of the solvePuzzle method, which returns a Hint object. If the grid is not completable in its current state, the X value of the Hint object will be -1. In grid generation, this is used to signal that the randomly generated grid is not finishable. In hint generation, this means that no possible hints can be generated. This triggers a message to the user, which appears on the hint button itself (see figure 5) informing them that they need to unmark some tiles for a hint to be generated. If the grid is finishable, the solvePuzzle method creates an ArrayList of the tiles it has set to get to a finished state, and then picks one from the list at random to return. The hint object returned is used to set the appropriate tile to a green version of the required symbol.

Figure Hint displayed in the Marupeke Grid

## 5.4 Quit Button

A quit button is implemented. When pressed, it triggers a popup window, asking the user to confirm they want to quit. If conformation is given, then Platform.exit is called, terminating the application. This makes it slightly easier for the user to exit the application.

## 5.5 Saving and Loading a Grid

The application offers the user the option of saving their progress mid game, and loading the game later, even after exiting and re-entering the program. The save button (see figure 5) calls a method of the controller called writeToFile, which in turn calls the models writeToFile method. This saves the grid in its current form to a folder within the project called SavedGame. When the load button (see figure 5) is pressed, it calls the gameGrids loadFromFile method. This sets the gameGrid equal to the grid returned by the method, the grid saved in the SavedGame folder. The action event for the load button also sets the new grids GUI to the gameGUI, and the Controller’s gameGrid to the new gameGrid. It also increments an IntegerProperty field in the controller called gridChange. In the start method of the GUI, a listener is attached to this integer property. When it changes, the root node is changed to reflect the new grid, and the primary scene is set to this scene, and the show method is recalled. This completes the functionality of loading a saved game. This is a useful feature for the user, as it means they do not have to finish a puzzle in a single sitting. They can save it and come back to it later.

## 5.6 Music and Sound Effects

The application features music and contextual sound effects. These are implemented using JavaFX’s MediaPlayer objects. They are initialised in the init method. The music and sound files are saved in folders within the project called Music and Sounds respectively. Using the file path, they are added to the MediaPlayer objects. The music is set to play in the start method. It is also set to restart upon ending, allowing it to repeat indefinitely until its stop method is called. This is called inside the action event that triggers the end of the game. Here, another MediaPlayer plays a sound effect to indicate the game has been won. A trumpet fanfare sound effect is of course appropriate for this occasion. It is called to play in the same way as the music, but as it does not need to repeat, an action event doesn’t need to be set for when it finishes. The three other contextual sound effects are a buzzer sound effect for a violation, a choir sound effect for the hint button, and a click sound effect for all other buttons. These are set to play in the action events for the buttons. All of the audio files are licenced under Creative Commons copyright licenses. This means that they are free to use, as long as appropriate attribution is provided (Creative Commons, n.d). The required attributions are present at the bottom of the application. Though they do not alter the gameplay, music and sounds offer a subtly more enjoyable experience to the player.

Figure Buttons for Hint, Save, Load, New Game and Quit from the right-hand panel of the Marupeke game

## 5.7 Loading Popup

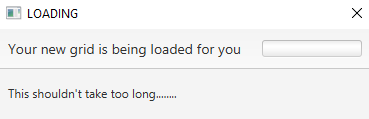
In the event that generating a new game takes a long time due to the size of the grid, a loading popup is implemented (see figure 6). This informs the user that the application is still running, as without it, it may appear as though the application has frozen. This feature has two bugs that cannot be fixed at this present time. The first is that the loading bar does not animate as it is supposed to. The second is that, occasionally and for no discernible reason, the text on the popup disappears. These issues however do not affect the functionality of the popup. The animated loading bar is merely for aesthetic, rather than serving as a real indicator of loading progress. The missing text is inconvenient; however, the popup title is still present when the text is missing. The title reads LOADING, making it clear what the popup is meant to convey.

Figure Loading popup from Marupeke game

# 6. Bibliography

Creative Commons, n.d. About the Licenses - Creative Commons. [online] Available at: <https://creativecommons.org/licenses/> [Accessed 10 April 2021].