**COMP1206 MathDoku Instructions**

**Guide:** This document will help us run and use your application during marking. Please complete the sections below. You may want to include screenshots if this helps explain the functionality. For most sections, 1-2 sentences are probably sufficient.

If you did not implement a particular part, please write “not implemented” in the relevant section.

These instructions are not assessed directly, but they will help ensure that we do not miss any important features of your application.

|  |
| --- |
| **Installing and Running the Application (Part 1)**  *Copy and paste the contents of your README.txt file below.* |
| To compile and run the application you need to compile and run the Launch class, which is in the MainMenu directory. |
| **Starting a Game (Optional – Part 1)**  *If any additional steps are needed to start a game, briefly describe them here.* |
|  |
| **Cell Completion (Part 3)**  *Describe how to enter and clear cell values by keyboard and by mouse.* |
| By Keyboard:  Clicking on a game cell will allow you to enter a number via the keyboard. You can delete the number using the backspace key.  By Mouse:  Clicking on a game cell will clear it from any input in it (if there is any). There is a number pad on the right side of the screen which allows the user to enter a number in a cell by:  clicking the cell they want to input in;  clicking on the number they wish to input in the cell; |
| **Can your application handle - and ÷ cages with more than two cells? (Part 4)** |
| Yes |
| **Mistake Detection (Part 4)**  *Describe how to enable mistake detection in your application.* |
| Click on the button called “Show Mistakes”. If there are mistakes it colours the wrongly entered numbers in red. To disable this mode, click on the same button again (it will rename itself to “Hide Mistakes”). |
| **Win Detection / Animation (Parts 4 & 8)**  *Describe how the application notifies the player when the game is won (including any animations you have implemented for Part 8).* |
| When you have won, a text that says “Great Job!” will appear on top of the game grid. The animation is that the text will continuously change its size. Clicking on the game grid will remove the winning animation. |
| **Clearing (Part 5)**  *Describe how to clear the board.* |
| Click the button “Clear”. An alert will appear asking you to confirm. After confirming, it will clear the grid. |
| **Undo/Redo (Part 5)**  *Describe how to undo / redo actions.* |
| After entering a number, the “Undo” button will become available to click on. After clicking at least once on the “Undo” button, the “Redo” button will become available to click on. |
| **Loading Files (Part 6)**  *Describe how to load puzzles both from file and through text input. Also mention any limitations in what puzzles you can load (if any), e.g., up to a certain size if smaller than 8x8.* |
| From File: From the Main Menu, click on the button “Load from File”. A file selector will open up and the user has to select a file from which to import a puzzle.  From Text: From the Main Menu, click on the button “Load from Text”. A new window will appear in which the user has to enter the text from which the puzzle will be loaded. When the user has entered the text, they must click the button “Done” so the puzzle can be loaded.  Limitations (optional): Can load up to 9x9 included. |
| **Font Sizes (Part 7)**  *Describe how to change font sizes* |
| Click on button “Change font” will make a new window appear in which the user can select a font they want. After clicking on a selected font, the window will close automatically and the game font will be changed to the selected one. |
| **Solver (Part 9)**  *Describe how to solve a puzzle, how to get a hint and any limitations there might be (e.g., up to what size you can solve reliably and within <1 min). Also mention where we can find your code for solving the puzzle (which files and lines)?* |
| not implemented |
| **Random Game Generator (Part 10)**  *Describe how to generate a random game, including what options the player can select. Also specify where we can find your code for generating the puzzle (which files and lines)? Where in the code do you ensure there is only one solution (which file and lines)?* |
| not implemented |
| **Additional Information (Optional)**  Any other information that may be useful for us to know. |
| In order to be clearer I have made a Main Menu, from which the user can navigate through the different ways to play the game. There is an option in the Main Menu to load an example puzzle. This option generates the same puzzle every time. The other two options in the Main Menu allow the user to load a puzzle from a file, or from a text. All the puzzles have the same functionality (it doesn’t matter if you load them from a file, or from text or if you play the example puzzle).  Like the Main Menu, I have added other functionalities to the project (such as the button that shows the game rules). But the biggest modification to the rules is that instead of having bold borders to differentiate the cages, I have used different colours. In my opinion, adding colours, makes the interface more pleasant to look at, while doing the same job as just having bold borders. If there are similarly coloured cages next to one another, there is an option to change the colours of all the cages (button “Change Colours”).  Also, the project has been written in IntelliJ, and if you read the code there are a lot of comments named “region”. They are a way to collapse big amounts of code and are very useful. But these “regions” are not collapsible in some other development platforms (NetBeans) and they just look quite out of place if they aren’t run in IntelliJ. |