COMP1406 - Assignment #5

(Due: **Mon. Mar 12th @ 12 noon**)

In this assignment, you will build a slider puzzle game from scratch. The purpose is to become familiar with event handling for various window components.

(1) Slider Puzzle Game

You will be creating the following application:

This application represents a slider puzzle game. There are 15 tiles that can be exchanged horizontally or vertically with a blank tile in order to shuffle the image pieces around. When the tiles are arranged in the correct order, they form a complete image and the game is done. The game is timed ... so the faster you can complete the puzzle, the better your score.

The game must meet all of the following requirements:

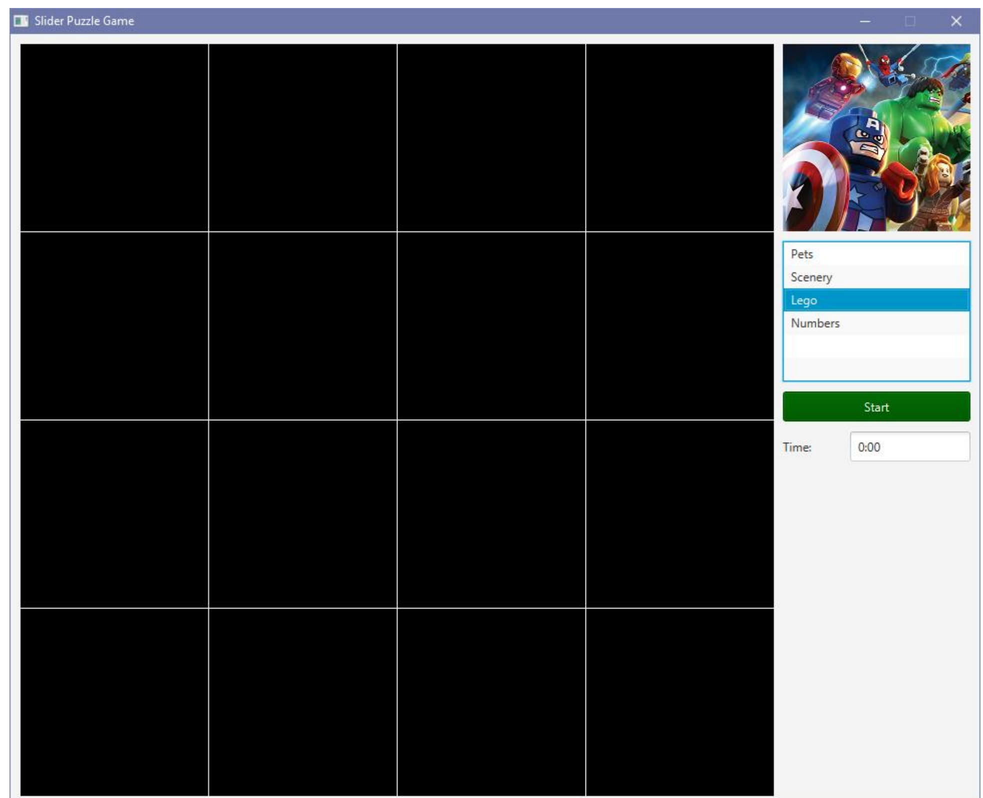
- There is a 4 x 4 grid of 16 Buttons that will hold the images for a puzzle. Your interface should show these buttons as indicated in the screen snapshot above. Each button should be 187 x 187 pixels in size. With a 1 pixel spacing vertically and horizontally between each button. You will need to set the padding to 0 for each button using `setPadding(new Insets(0,0,0,0));` This will allow the image to take the full space in the button. There are 4 sets of images available for you to use (although you can add your own as well). These are available on the course website, where you downloaded the assignment. All images must be unzipped and copied into the **src** folder of this assignment in order to be able to be loaded properly. For each puzzle, there are 17 images ... one is a thumbnail image which shows the whole image in 187x187 size. The others are the individual tile images (each 187x187 in size) and are labeled with the format Name_RC ... where Name is the name of the puzzle (i.e., Pets, Scenery, Lego or Numbers) and R is the row number and C is the column number for that image (i.e., 0, 1, 2 or 3). Recall that you set the image for a button as follows:

```
setGraphic(new ImageView(new Image(getClass().getResourceAsStream(filename))));
```

  where `filename` is a string representing the name of the file (e.g., "Lego_21"). It may be a good idea to make sure that you can create the buttons and display all of the images from a puzzle in order on each button before you go any further on the assignment.

- Your GUI should also show the complete image (i.e., use the available **Thumbnail** image) as a **Label** and underneath it should be a **ListView** showing (at least) the names of the 4 puzzles. See snapshot above.

- There should be a **Start/Stop** button (shown as **Stop** on the snapshot above) as well as a "Time:" **Label** and a **TextField** that shows the time that has elapsed since the puzzle was started. All components must be "nicely" arranged/sized with reasonably consistent margins all around.



- Upon window startup, the buttons should show the **BLANK.png** image on each button, the thumbnail **Label** should be enabled and the **Start** button should be **DARKGREEN** with **WHITE** letters indicating "**Start**". The time should be "**0:00**". See image here.

- When an item is selected from the list, the appropriate image should be shown in the Thumbnail **Label**.

- When **Start** is pressed, the game should begin running. When the game is running, the following should happen:

    - The Thumbnail **Label** should be disabled (it will look faded as in the first snapshot on this assignment).
    - The **Start** button should become **DARKRED** and indicate "**Stop**" instead of "**Start**".

    - The images on the buttons should be changed to the 16 images for the tiles that make up the puzzle image. One of these tiles (chosen randomly) should remain as a blank one. The tiles should be shuffled randomly (see first snapshot) ... more will be mentioned about this below.

    - A **Timer** should be started. Use a **Timeline** object (from the **javafx.animation** package). Here is how to set up the timer to tick once per second. You should set this up in the **start()** method:

```
updateTimer = new Timeline(new KeyFrame(Duration.millis(1000),
                                new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        // FILL IN YOUR CODE HERE THAT WILL GET CALLED ONCE PER SEC.
    }
}));
updateTimer.setCycleCount(Timeline.INDEFINITE);
```

    To start the timer, you use: `updateTimer.play();` This should be done when the **Start** button is pressed. When the **Stop** button is pressed, you should stop the timer by using: `updateTimer.stop();`

- At any time that the game is running, the time should be displayed in the **TextField**, with proper formatting. That is, it should show the number of minutes that have elapsed, followed by a '**:**' character and then the number of seconds that have elapsed since the last minute (always shown as 2 digits). You should use **String.format()** here. You'll have to keep track of the number of seconds that have elapsed since the timer was started.

- The game should stop if the user pressed the **Stop** button, or when the puzzle has been completed (more on this is described below). At this time, the timer should be stopped (use: `updateTimer.stop();`). Also, the thumbnail **Label** should be **enabled** again. The **Stop** button should become **DARKGREEN** and indicate "**Start**" instead of "**Stop**". If the user pressed the **Stop** button, then all 16 tile buttons should show the BLANK.png image. If the game ended due to tile completion, then all 16 tile buttons should be **disabled** and the blank tile should be replaced by the puzzle image that was missing from the start (you will want to remember which one was replaced by the blank image upon starting so that you can show it at this time).

- During the game, when the user clicks on a tile button, you should do the following. Determine which **row** & **col** was clicked on (an example of this was given in chapter 5 of the notes with our **ToggleButtonsApp**). Then you should look **above**, **below**, **left** and **right** of that **(row, col)** location to see if the blank tile is there (be careful of boundary cases). If the blank tile is there, then the image on the blank tile should be swapped with the image of the tile that was clicked on. Visually, it should appear that the clicked tile slides into the blank position, moving the blank position into its position. It would be a good idea to create a method called **swap(row,col)** that takes the **(row,col)** of the button that was clicked on, and then attempts to find the blank image around it and swap it.

- To shuffle the images upon startup, it is good to start with the full 16 images, and then simply call your **swap(row, col)** method 5000 times or so. This will cause a bunch of tiles to be slid around and should give you a reasonable shuffling. Then replace one of the tiles at random with the blank one. Remember to keep hold of the replaced image so that you can show it at the blank spot when the puzzle is completed.

- To determine when the puzzle is complete, you will need to know when each tile piece is in the correct location. That is, image "xxx_00.png" should be at the top left with image "xxx_01.png" beside it, image "xxx_10.png" below it, etc... It may be easiest to keep three 2D arrays. One for the buttons, one for the images of each button, and one for the coordinate of each button (i.e., the row/col of the image for that button). A coordinate can be stored using a **Point2D** object which has an **x** and **y** coordinate. You can determine when the board is complete by examining each button row by row and col by col (using the double FOR loop) and just check that the coordinate value matches the **(row,col)** of that button. This will only work if you swap the coordinates along with swapping the images in your **swap(row,col)** method. You do not have to do things this way, it is just an idea. If you have a simpler way to determine when the board is complete, you can do that instead.

- Before you hand in your assignment, make sure that you are handing in ALL of the images along with it ... in the same **src** folder. You will lose marks if any image files are missing.

- Have fun with this game. You may add features (no bonus marks though) such as additional puzzles (**DO NOT USE ANY INAPPROPRIATE IMAGES**) or perhaps a **TextArea** or **ListView** on the window that shows a sorted list of "high scores" (lowest time is the highest score).

IMPORTANT SUBMISSION INSTRUCTIONS:

Submit your ZIPPED **IntelliJ project file** as you did during the first tutorial for assignment 0.

- YOU WILL LOSE MARKS IF YOU ATTEMPT TO USE ANY OTHER COMPRESSION FORMATS SUCH AS **.RAR**, **.ARC, .TGZ**, **.JAR**, **.PKG**, **.PZIP**.

- If your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

- You WILL lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.