Trump Stumpers

By: Tin, Brian, Thomas

Problem

A point and click game was to be created. In the game, the player has 20 tries to click on a moving picture of Donald Trump and the player earns points each time they click on the picture. The speed and difficulty of the game depends on the mode chosen by the player: easy, medium, or hard. A high score table at the end will display the top three scores of the chosen mode.

(This is a political satire game. We thought about implementing the game based on shooting Trump with a gun but for school purposes, we decided to do Anti-ads instead).

Analysis

The project is created with a heavy emphasis on Javafx and event driven programming. Buttons are used heavily in this project as a source object. A lot of research was spent on the Animation class and Timeline, where both were used to have the Trump picture to move. Rather than using the timer class, we created our own timer which used timeline (with eventhandler).

At the beginning of the project, the way we implemented the events was extremely sloppy. Everything was forced into the start method, where we initialized all of the objects, and ArrayLists and etc. Difficulty settings were all forced into the start method as well. Data were all public, thus allowing user manipulation of the object's' data. The encapsulation and readability for other programmers was extremely poor. The program was also based on using Pane instead of StackPane, thus making all the labels and button were off coordinated. Also some of the methods were written as anonymous inner class, which could have been simplified with lambda. A lot of memory was also wasted as well, as nodes' visibility was switched to false when no longer needed. The correct approach would be to remove the node from the pane.

Later on the mistakes were mostly fixed. Methods were created to improve readability of the program, and to improve encapsulation. Objects were initialized outside of the start

method, and also created as private (for visibility/encapsulation purposes). We rearranged all the previous methods into lambda, and created separate methods for different functionality (such as different difficulty, lost, etc). The usage of Stackpane over pane transformed the appealing of the program, as our nodes were aligned better. Originally, we were going to use an PriorityQueue to store our high score, however it did not work. Despite researching on the PriorityQueue, we could not find a solution to the problem. We later on used to ArrayList, which ended up working. One improvement could be made into the program is how the high scores don't store after you close the program. One way to change that is to store files into the Arraylist itself, which then can store high scores for an indefinite amount of times the user closes the program.

Design

1. Starting from the beginning, many private/instance variables are created as their scope extends to every method/class within the Main class. All of these variables are used in several different methods and classes within the program.
2. All menus/pages are placed within their respective methods, and are called when a button/mouse click that leads to that menu/page is clicked.
3. Most texts and buttons are placed either in an HBox or a VBox, so that they can be both be placed in the center and can specify the space wanted between each text/button.
4. Every button or click has a similar event, which is to remove all/most current objects on the screen and call a method that will create a new , with the exception of the background and background music, which has their own options to remove them.
5. **Start Menu**
    a. The startMenu method includes a title, which uses a Text object, placed at a particular location, with a font size obtained from an interface, FontSizes, which we created.
    b. Three buttons are created in the start menu with the purpose of either starting a new game, which will bring you to a new menu that asks what difficulty does the user want to play, bringing up an instruction menu, or changing some options.

c. One more option on the start menu is the mute button, which either turns on/off the volume without having to go to the options menu.

6. **Instructions**
    a. The instructions page is mostly text, with a button that returns the user back to the start menu.
        i. All text is created with Text objects and combined using a TextFlow
7. **Difficulty Menu**
    a. There are three buttons placed in the center for choosing difficulty (easy, medium, or hard), which each controlling the animation, or timeline, in a different way. All options will change the update, or speed, of the Trump head, start the timeline, and start the game.
    b. There is a fourth button, which is the same as the instruction's button, which brings the user back to the start menu.
8. **Options Menu**
    a. The options menu has four options; return to start menu, mute, remove/re-add background, or change image for the game.
        i. Start menu is the same as the other return to start menus, it removes all elements currently on the screen and calls the start menu method to add start menu elements back on.
        ii. Mute controls a boolean which forces all sounds off.
        iii. Remove/re-add background either sets the background as null or sets it back to the flag.
        iv. Change Image either switches the Trump head with a chicken to avoid political elements, or, if the Trump head is prefered, can be used to swap it back.
9. **Gameplay/mouseClickedToPlay**
    a. Depending on which difficulty the user chose, different algorithms were set for the Trump head, or chicken, to follow. Two texts are placed onto the screen,

which are the score count and the number of tries left, which are recalled after each click to update them.

i. In the hard difficulty, a timer is added in the top center.

b. After the game ends, all elements except for the score count is removed and the lost method is called to bring up the gameover screen.

c. Sounds are placed into a HashMap for clarity. Depending on whether you clicked on the head or on the pane, it will give you a booing sound or a cheering sound. Whenever you click again, the program will check if any sound is playing using the for each loop and a set containing the keys for the sounds.

d. For our program, instead of using the image as the target for clicking, we decided to use the pane for the event. The pane takes the x and y coordinates, which is the upper left corner, to the bottom right, defined by us, as valid points, everything else is an invalid, or a miss.

10. Game Over Menu

a. Other than text, there is a scoreboard for each difficulty. Using a priorityqueue, all scores are stored into this queue, in reverse order, and is given the respective difficulties' score. Three is removed, unless the number of scores is less than three, then it forces a break, change the rest of the scores to N/A, and re-adds them back into the queue.

11. Other Methods

a. Creating a high score list

i. In the data field, an arraylist that consists of three PriorityQueues that should output numbers in reverse order, that is, from highest first. In the lost menu, a text array of size three is created and is used to show the highest scores. In the lost menu, depending on the difficulty, adds the score into one of the PriorityQueues and then passes that queue and the text array into this method. The way this method works is that first, an integer that holds what index of the text array the program should input at the moment, and an arraylist to hold numbers from the PriorityQueue is

created. The first for loop starts that mentioned integer at the first index, then checks if there are anymore elements within the PriorityQueue, and if there isn't, forces the current for loop to end. If there are still elements, add the score to the arraylist of integers, which should, since it is a PriorityQueue in reverse order, give its greatest integer, then add a text of that number into the text index. In the next for loop, if the program ends due to next enough numbers, the next for loop starts at the index the previous loop ended at and puts the rest of the text indices as N/A. The last index adds back into the PriorityQueue the integers it put into that array.

b. mouseClickForMuteImage

i. The mute button in the start menu has two functions, to mute the game, and to change its image from mute to has volume. Method gives the image this event.

c. removePaneEvent

i. When the game ends, we have to remove the pane's event, thus we call this method to make the mouse click event empty.

Implementation

Libraries used: JRE System Library, JavaFX SDK

Tools used: JavaFX, Eclipse, Google Drive, E-mail, GitLab

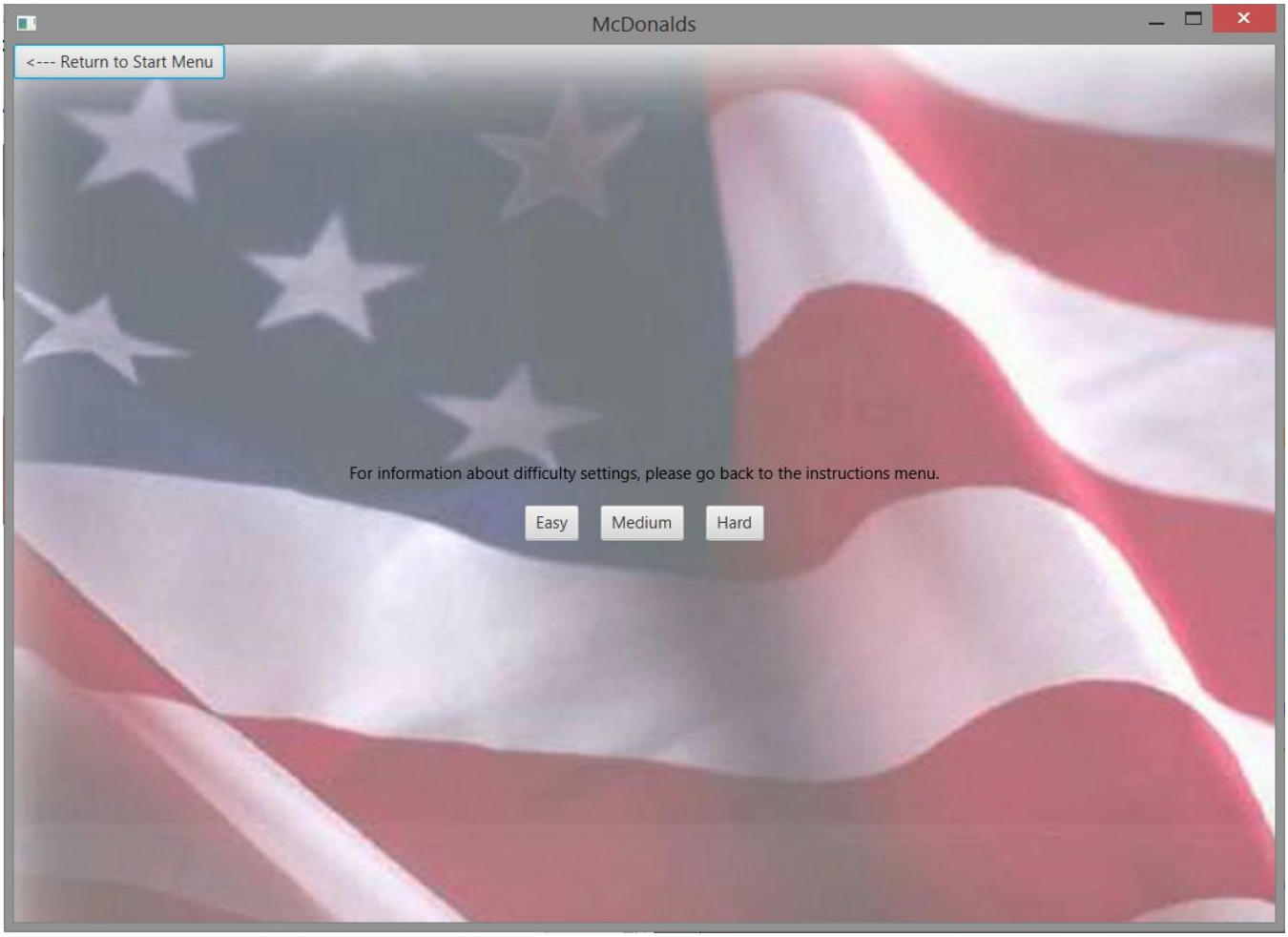
In the early stages of the project, we exchanged code through Google Docs and also through email in the form of .zip files. Later on, in order to update the project, we fetched and pushed code through GitLab.

Class Topics used: Class Definition, Inheritance, Abstract Classes, Interfaces, Generics.

PriorityQueues, HashMap, JavaFX

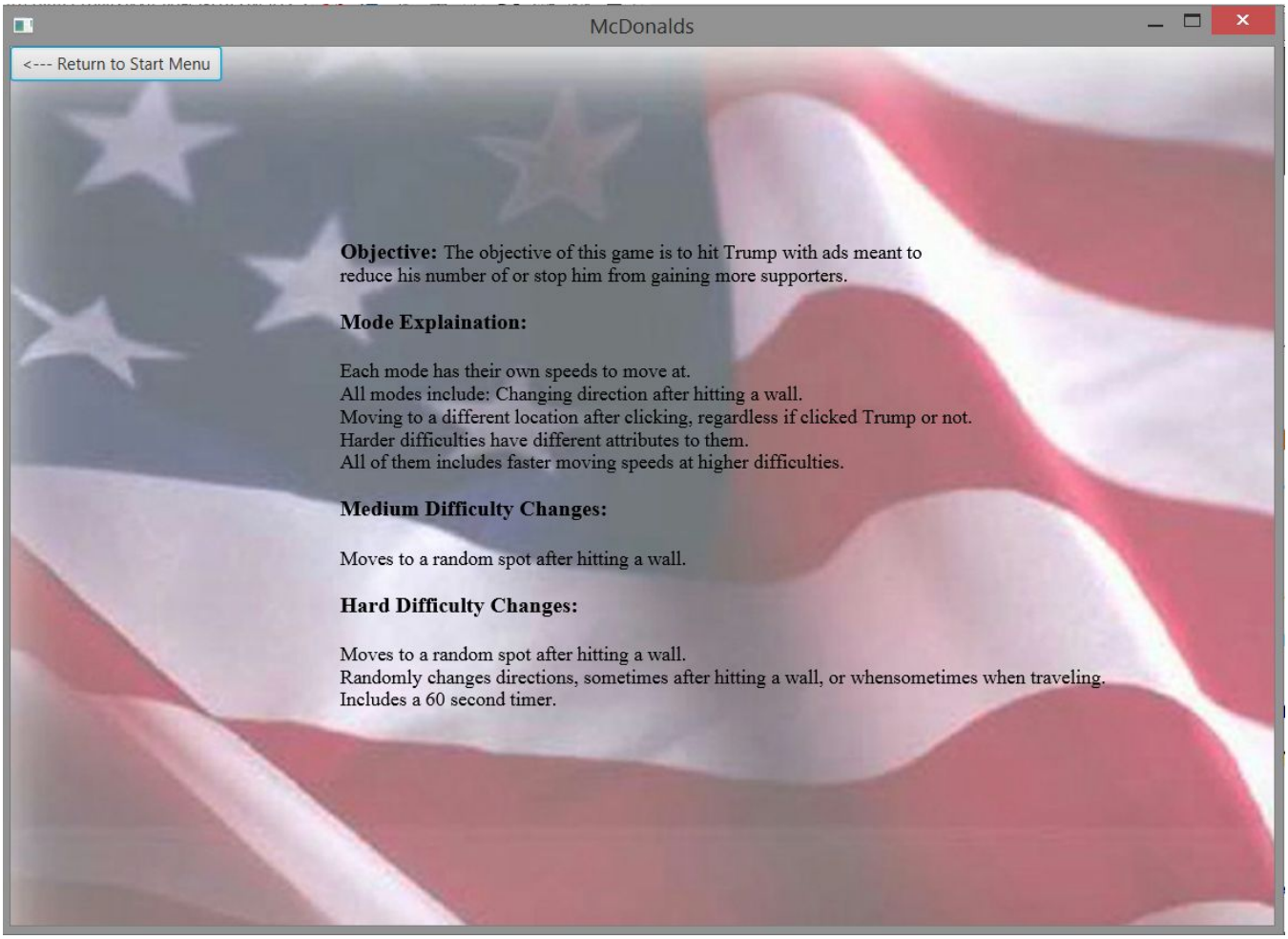
Evaluation

The main menu, where the title is displayed with the New Game, Instructions, and Options buttons.



There are three difficulties that the user can choose, which are easy, medium, or hard. There is also a return to start menu button at the top left.

An example of end game high score (separate high score list for different difficulty), with accuracy at the end.



A screenshot of the instructions (easy is omitted due to how “easy” it is).