Contributors: Massi, Brevin, Rachel

Whac-A-Mole's Documentation

User Manual

1. Upon starting the game, you will be prompted with whether or not you would like to restore a previous saved game. If this is your first time playing the game or you have not previously saved a game, please enter "n". If you have previously saved a game, you may choose to enter "y" or start a new game by entering "n".

2. The game will then ask you to input, in the following order, the number of rows of Magic Holes you would like, the number of columns of Magic Holes you would like, your name, and how much time you would like (in seconds) between the Magic Holes changing color on their own. (Please enter positive integers for the number of rows, columns, and seconds.)

3. The game screen will then open with many black Magic Holes and one red Magic Hole. The goal of the game is to click the red Magic Hole and gain points, eventually leveling up every 50 points you obtain. Upon leveling up, you will hear a sound and see your level has increased. Depending on your level, the amount of time between the red Magic Hole changing spots will decrease and the amount of points you will get for every click on the red Magic Hole will increase.

4. After clicking five non-red Magic Holes, the game will end, giving you your final score and level. You may press enter at this time to start a new game.

5. Clicking the Hall of Fame button will show you the Hall of Fame. Here, the top 16 scores and names of people who have achieved them are shown.

6. Clicking the Mute button will mute the game sounds. If you would like to unmute them, simply click the button another time.

7. The Save button will save your current game and enter your name to see if it will be displayed in the Hall of Fame. If you would like to restore from your game in the future, simply enter "y" when prompted to at the start of the program.

Contributors: Massi, Brevin, Rachel

Non-functional Features

Currently, we have no features that are not working properly. While there may be features that could be added in the future, nothing we have implemented currently is inoperable.

Enhancements and Reductions to the Game

We have made small adjustments to our program to allow for some enhancements. We have adjusted our Mute Button to allow switching back and forth between a muted state upon pressing. Additionally, we have also made our Save Button more efficient by saving to both the Hall of Fame and the Restore State upon being pressed. The only Reduction we have made is removing our plans for a larger Full History to be implemented. While the full history can still be viewed in the "hall_of_fame.txt" file, it is not able to be viewed in-game.

Roles and Contributions

Throughout this project our team worked very well together, and strived to evenly distribute the workload for each new phase. For Phase Zero and One we all got together to work out the bulk foundation of our program together as a team. Following Phase One, we created a system to evenly divide the requirements of each subsequent phase. We met once early in the week to go over each requirement and give each member their role. For example, Phase Three's requirements included adding sound upon correct and incorrect clicks, adding a timer, and thoroughly documenting our progress. Seeing a clear three parts, we divided the roles, with Brevin completing the sound, Massi completing the timer, and Rachel completing the documentation. After this, we met at the end of each week to consolidate our code.

Sound Libraries

Java io InputStream: The input stream is used to create a path to the particular sound file that we are referencing for successful, unsuccessful, and level up sound effects.

Sound Clip: Clip is used to create a clip object which is capable of playing the input stream.

Sound AudioSystem: Audio system is used to feed the input stream into the clip object.

Contributors: Massi, Brevin, Rachel

Audio Implementation

To create a functioning audio aspect for our Magic Holes game we created a playSoundEffect function. This function first implements an input stream, which creates a path to the particular sound file that we are referencing. Next, the function creates a clip object, and then uses audio systems to feed the input stream into the clip. Finally, it uses the clip's start method to play the audio clip. After this function was created we used it in our if-statements that determine whether a successful or unsuccessful click has been committed.

Real-Time Implementation

For handling the timer of the application, we used the Timer library from Java Swing. The timer was initialized with two parameters: the interval (in milliseconds) and the actionlistener to be called upon the lapse of time. The interval, in our case, was taken through user input and communicated with the Timer. The actionListener for our Timer was simply invoking the colorRandomization function which was in charge of picking a random ellipse from our ArrayList of ellipses and coloring it.
Some important and useful features of Timer that were really helpful for us were: timer.start(), timer.restart() and timer.setDelay(). Timer.start() was called in the constructor so that it starts timing. The restart feature was called upon a successful click, for instance. SetDelay was called upon levelling up.

Levels-of-Play Implementation

Our idea of Levels of Play was implemented in three different aspects: the changing of a score multiplier as levels increase, the shortening of a timer as levels increase, and the increasing of levels itself. Starting with leveling, upon the MouseEvent detecting a successful click on an active ellipse, an if statement detects whether or not the current score is now divisible by 50 with no remainder and that the current score is not 0. If the score meets these criteria, the level counter is increased by one. Additionally, the score multiplier variable, which is used to calculate how many points the player earns after clicking an active ellipse, is increased by one, as well. The timer interval entered by the player is then divided by two and set as the new interval. A new timer is then started with this new interval. A sound is then played to indicate leveling up for the player, and the level_result and score_result variables are updated in order to update the display for the user.

Hall of Fame Implementation

Contributors: Massi, Brevin, Rachel

The Hall of Fame lists the top 16 players with the highest scores. The table containing the scores and each player's name can be displayed upon the user's click on the 'Hall of Fame' button. In order to create the table displaying this, we used JTable. However, we used DefaultTableModel from Swing Table in order to add rows and columns first. Then, the DefaultTableModel was contained inside the JTable. Additionally, we used Scanner to read the text from the file and then put them into ArrayList, followed by mapping names to their scores.

Although the Model added the rows with the given data, the data was sorted in descending order using collections.Sort() and custom Comparator. However, the challenge still remained for how to maintain the order in a map, which is why we used LinkedHashMap. LinkedHashMaps maintain the order, can be iterated in format of <K, V> where K is key and value is V, and they are also efficient.

Saving and Restoring Implementation

Our saving and restoring is done entirely through ArrayLists and writing to text files using BufferedWriter, PrintWriter, and FileWriter. Using the libraries for each of the writers, upon the press of the Save button, which is detected by the usage of a MouseListener EventHandler, the program writes the current name and score of the player to the "hall_of_fame.txt" file (which is used in adding data to the table mentioned above) and also writes the current name, score, level, lives, score_multiplier, interval, number of rows, and number of columns in the current game to the "current_game.txt" file. In order to restore the game, when starting a new game, within the console, the user is prompted with the question "Would you like to restore from the last save game? (y/n)". If the user responds with a yes, the program checks to see if the "current_game.txt" file exists (and returns an exception if it does not) and then pulls the data from the file using a Scanner object delimiting between objects on a semicolon. The program then stores this data within an ArrayList, and then sets each variable to these values, converting where necessary.

Persistent Information

The information that was taken from the user and the game was saved in two text files: "hall_of_fame.txt" and "current_game.txt". "hall_of_fame.txt" stores the player's name and score upon the player's click of the Save button. Then, this information will be available if the user is listed in Hall of Fame. Additionally, when the player clicks on the Save button, that specific game's entire state including their name, score, level, lives, time interval, rows, and columns are all saved in "current_game.txt".

The next time, if the player wants to restore the game, he/she can state it during user input and the game can be picked up from where it was left off.

Our Design Patterns

In order to achieve our goals, we had two classes: HolesComponent and HolesFrame, excluding HolesMain which contains the main method. HolesFrame is where we set up the layout to BorderLayout and positioned all components in their places. Additionally, that is where we handled the buttons, Save, Mute and Hall of Fame, as well as the JTable which displays the top 16 players.
Whereas, HolesComponent is where most of the program happened when it comes to the way the game is played and looks. Here we set the default size when the user opens the game. Also, here we take the user input which leads to the number of ellipses in rows and columns and the interval of seconds. Among all the methods, the most important one was paintComponent, which took care of drawing the ellipses and random coloring ellipses, besides showing the level, lives, and scores while changing them upon other events. Additionally, this is where the mouse click events were handled and different sounds were played upon different events.