Simon SAYS

Nate Agcaoili

CSCI 4370

Handheld/Ubiquitous Computing

Dr. Hong Zhang

Table of Contents

Title Page …………………………………………………………………………… 1

Project Description ……………………………………………………………… 3

Game Logic ………………………………………………………………………… 3

Activities …………………………………………………………………………… 7

Actors and Scenario …………………………………………………………… 7

State Chart Diagram …………………………………………………………… 8

Application Wireframe ………………………………………………………… 9

Tools and Technologies ……………………………………………………… 10

Source Code ……………………………………………………………………… 10

Conclusion ………………………………………………………………………… 11

Project Description

Simon Says is a round-completion based memory game that gets increasingly more difficult as the user progresses. When the game begins, you will be greeted with a 3 by 3 grid of buttons. Simon will then proceed to indicate which button he would like you to press by turning the button from blue to white. In order to move on to the next round, you simply have to push the same button that Simon did. The following rounds after, Simon will continue to add a new button to his sequence one round at a time until you are no longer able to recall the exact sequence that Simon provided.

The application uses SQLite to store score information such as the highest score the user has ever achieved, the average score of all the games the user has played, and the most recent score that the user earned. These scores can be viewed and reset in the Scores Activity.

Game Logic

Variables - Simon Says consists of many variables that work behind the scenes to store various information. Below are listed those that are most integral to the game's core mechanics.

- **running** [Boolean] – determines whether the game is running (True if the user hasn't failed yet, False if the user has).
- **playerTurn** [Boolean] – determines if it is the user's turn to repeat the sequence or Simon's turn to display his sequence.

- **currentRound** [Integer] – keeps track of what round the user is on, automatically increments when a round is completed.
- **gameButtons** [Array of Buttons] – stores the nine game buttons in ascending order.
- **simonSequence** [ArrayList of Integers] – at the start of a new round, a random number between 0 and 8 is generated and added to simonSequence. This integer will correspond to an index of gameButtons.
- **playerIndex** [Integer] – when it is the user's turn, playerIndex is reset to zero at the start of each round. It automatically increments each time a user clicks on a button. The program checks to see if the button that the user clicked matches the button in the playerIndex index of simonSequence.

User Interface – The UI for Simon Says Game Activity was designed using the Design builder for XML files in Android Studio. The main components that the user sees are listed below.

- **butOne, butTwo, …, butNine** [Buttons] – these are the nine buttons that consist of the 3 by 3 grid that Simon and the user interact with. Due to it being Simon's turn at the start of the game, all 9 of the buttons are disabled by default.
- **turnDisplay** [TextView] – displays whether it is Simon's turn or the user's turn. The text is briefly be set to "Correct!" once the user successfully completes a round.
- **roundDisplay** [TextView] – displays what round the user is currently on. The text is set "Score: [roundNumber]" once the user fails to complete a round.
- **playAgainButton** [ImageButton] – hidden by default, appears once the user fails. If pressed, a new game is started.

- **mainMenuButton** [ImageButton] – hidden by default, appears once the user fails. If pressed, the user returns to main menu.

Main Functions – Simon Says operates using a series of functions that interact with the different functions and UI components to produce a playable game. Listed below is an example of functions that are called as the user progresses through the game.

- **onCreate()** – initializes all predefined variables once the activity is first launched then calls playGame()
- **playGame()** – is recursively called through other functions throughout the game until running is set to false.
    a. If the game is no longer running, gameOver() is called.
    b. Else if it is Simon's turn (playerTurn == false), roundDisplay is set to currentRound, a new number is added to simonSequence, and the sequence is displayed to the user. Once the sequence display finishes, playerTurn is set to true, all the buttons are enabled for the player to interact with via toggleButtons(), and playGame() is called.
    c. Else if it is the user's turn (playerTurn == true), the user is tasked with clicking on the game buttons in the same order that Simon displayed. If the user successfully recreates Simon's pattern (playerIndex == currentRound - the user has reached the end of simonSequence), playerTurn is set to false, all buttons are disabled via toggleButtons(), playerIndex is reset to 0, currentRound is incremented by 1, the user is informed that they are correct, and playGame() is called.
- **gameButtonClick()** – All buttons are set onClick to this function which when they are enabled, verifies that the button that the user clicks matches the same order that Simon displayed. This is

achieved by checking if the button that was clicked is equal to the grabbing the gameButton index of playerIndex of simonSequence.

- **gameOver()** – scoreboard SQLite database is updated based on the user's score, game over screen appears, and Play Again and Main Menu buttons become available.

Helper Functions – The functions listed below are used to complete minor operations that keep the game transitioning between different states in a seamless manner.

- **toggleButtons()** – toggles whether or not the game buttons are enabled based on whether or not it is the user's turn.
- **setActivityBackground()** – changes the background of the activity based on the state of the game.
- **addToSequence()** – generates a random number between 1 and 8 and adds it to simonSequence.
- **updateScoresDatabase()** – updates the scores in the SQLite database scores table.

Multithreading – Due to Android UI View objects not being thread-safe, I had to utilize AsyncTask when adding delays between changes in to the UI. Below are classes that extend AsyncTask that I wrote to perform such changes to the UI.

- **DisplaySequence** – calls addToSequence(), iterates through simonSequence and publishes the current button in simonSequence to be displayed in the main thread. Once the thread has been completed, playerTurn is set to true, the user is notified that it is their turn, and playGame() is called.
- **RoundComplete** – the screen and all buttons briefly become green to show the user that they have successfully completed a round. Once the thread has been completed, turnDisplay is set to "Simon's turn" and playGame() is called.

Activities

MainActivity – Serves as the main menu that users are greeted with when they start the application. From this screen, you can navigate to a new game, the scores page, or the about page.

GameActivity – All of the game logic is housed.

ScoresActivity – User can view and reset their scores.

AboutActivity – Contains various information about the application and the development process behind it.

Actors and Scenario

Actors: User & Existing Software

Scenario Name: playSimonSays

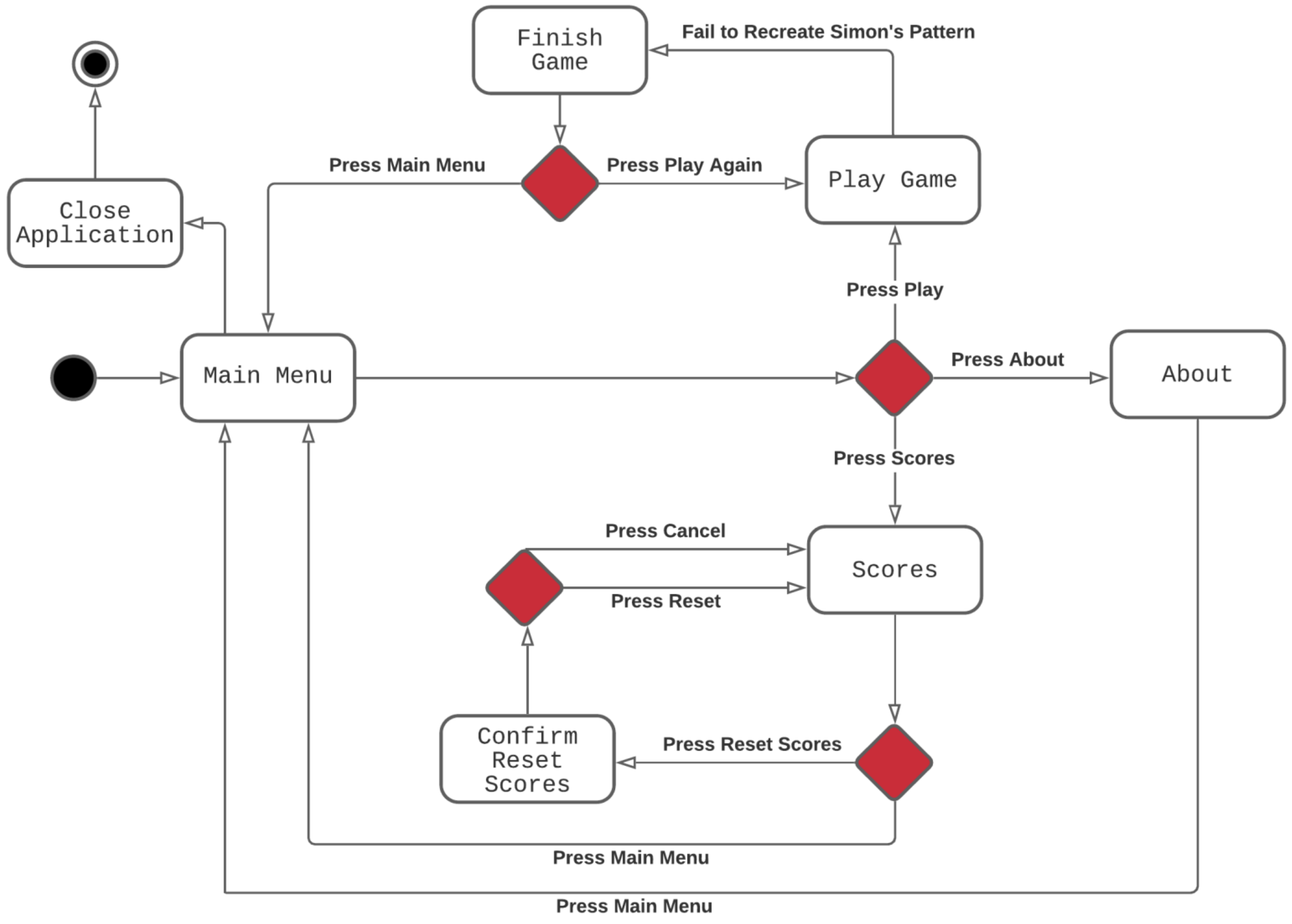
Participating Actor Instances: Nate: User, Simon Says App: Existing Software

Flow of Events:

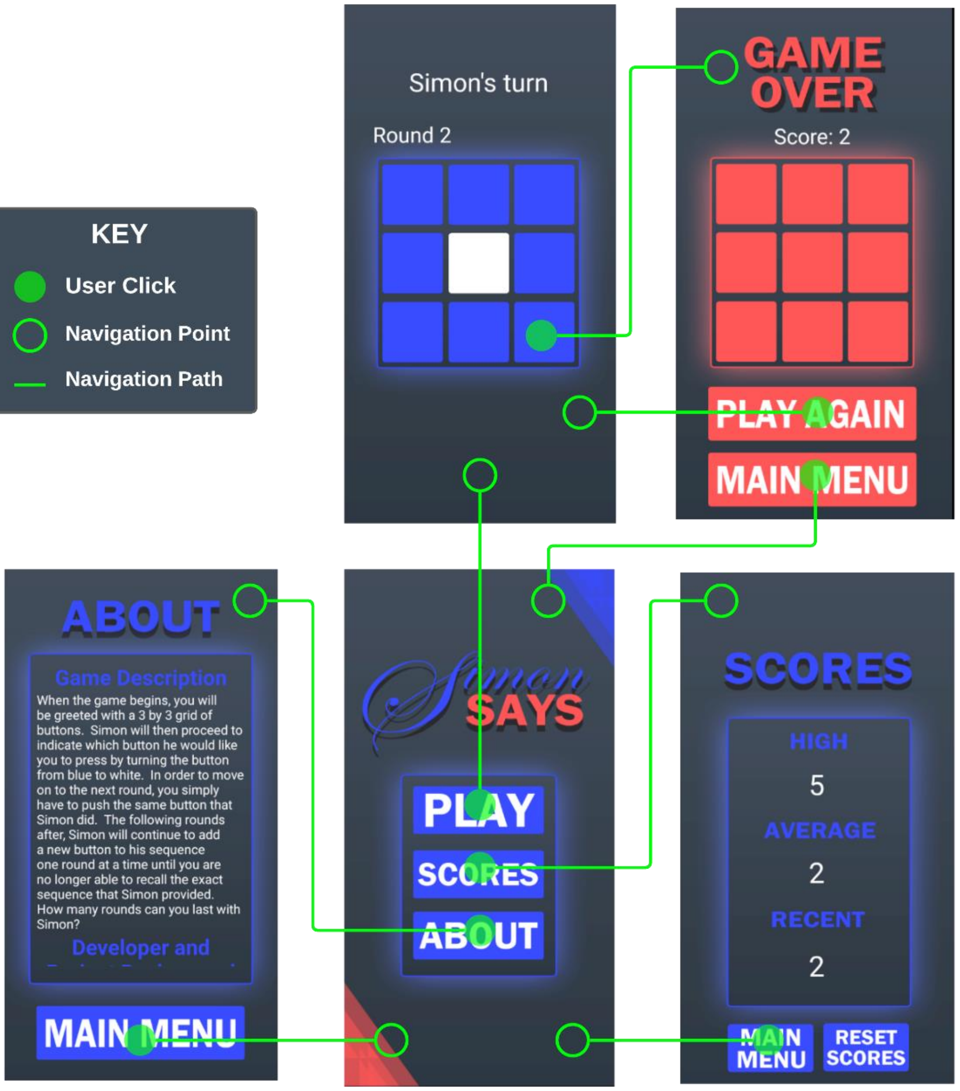
- Nate, starts Simon Says App
- Nate, presses 'PLAY' button
- Nate, plays Simon Says and earns a score of 12
- Nate, presses 'PLAY AGAIN' button
- Nate, plays Simon Says and earns a score of 2
- Nate, presses 'MAIN MENU' button
- Nate, presses 'SCORES' button
- Nate, presses 'RESET SCORES' button

- Nate, presses ‘RESET’ confirmation button
- Nate, presses ‘MAIN MENU’ button
- Nate, exits Simon Says App

State Chart Diagram



Application Wireframe



Game Description

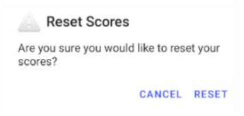
When the game begins, you will be greeted with a 3 by 3 grid of buttons. Simon will then proceed to indicate which button he would like you to press by turning the button from blue to white. In order to move on to the next round, you simply have to push the same button that Simon did. The following rounds after, Simon will continue to add a new button to his sequence one round at a time until you are no longer able to recall the exact sequence that Simon provided. How many rounds can you last with Simon?

Developer and

KEY

- ● User Click
- ○ Navigation Point
- — Navigation Path

If the user fails to reproduce the pattern that was displayed, the game over screen will appear.



Reset Scores

Are you sure you would like to reset your scores?

CANCEL RESET

When the 'Reset Scores' button is pressed, a confirmation box will appear verifying that the user wishes to reset their scores.

Tools and Technologies

Development Environment

Simon Says was developed using the Android Studio integrated development environment and Java as the primary programming language.

Assets

All images and audio files were original contributions by me. Images such as buttons and backgrounds were created using Adobe Photoshop. Audio files such as the button chimes were created using FL Studio.

Source Code

GitHub Repository

https://github.com/NateAgcaoili/Android-Simon-Says-Game

Conclusion

Overview

Completing this project has been a very rewarding experience for me. It served as a great opportunity for me to implement a lot of the Android application development concepts that I learned during this semester into an original project. In addition, I learned new aspects of development such as image buttons, activity backgrounds, and more.

Android Fundamentals Implemented

Through working on this project, I successfully displayed proficiency of Android application development fundamentals such as graphics, threads, and database usage.

What's Next

For the most part, I am very pleased with how my project turned out. The main issue that I encountered was with MediaPlayer objects being unreliable at times. For instance, if the Simon selects the same button two times in a row, the second chime will not play, but if Simon selects a button three times in a row, the third chime will play but still omits the second one. Similarly, if the user clicks on buttons with quick succession, it is likely that some chimes will be skipped. If I were to continue to work on this project in the future, I would try to figure out the source of these issues and implement a more reliable system when playing sound effects. Other than that, updates would mainly consist of minor UI or functionality changes.