MemTrain 1.0

Group 3

8/17/15

Table of Contents

Brain Storming

As a group, we were opting for an introductory level project with a fair amount of complexity. None of us have had any previous Android programming experience and so we wanted to pick a project that didn’t involve an internet connection or any advanced API due to the short timespan of the class and our lack of knowledge. We began by brainstorming a travel guide application to show users certain capitals of the United States and possibly weather and other various information. We dismissed this idea quickly because we thought it would be too simple, and so we are now opting to program a memory trainer game that gives the user an expanding and increasingly difficult pattern that they must follow. We feel the scope of this project is appropriate for our current knowledge of Android development, and it also incorporates a decent amount of complexity because we will have to design an algorithm to randomize a pattern and expand the pattern while keeping track of how the user is keeping up with the pattern on-the-fly. For example, the program may start by illuminating one of six shapes, and the user will be required to click the appropriate shape. Then the system will add another shape to the sequence and the user will have to keep up with the expanding pattern the system generates. Optimally, we would like the system to continue expanding the random pattern until the user makes a mistake, which will result in a termination of the game. While we are still debating on the specifics of the GUI, we feel we will be using a combination of shapes and colors in a 3x3 matrix array. We will also be adding a custom score system as well as a leaderboard to keep track of how well a user performs over the game’s lifetime. A difficulty level system will also be implemented with various levels of speed and combinations to test the user’s ability. We plan to stay in a 2D environment, but we will be implementing advanced animations to make the application more user-friendly. Our biggest challenge will be developing an algorithm to match the user’s pattern with the system pattern in the shortest amount of time possible so that the game maintains a high level of fluidity.

Software Requirements Specification

for

MemTrain

Version 1.0 approved

Prepared by Group 3

Comp 380

7/23/15

1. Introduction

1.1 Purpose

- The purpose of this project is to create, develop, and maintain a memory trainer application for Android. The application will come bundled with many extra utilities such as main menu, leaderboard, scoring, and help features. The main application and extra utilities are all part of the same subsystem and depend on each other for full functionality.

1.2 Document Conventions

- The higher-level requirements for this application are the same as its detailed requirements and do not require a different typographical convention than the one used on the main application.

1.3 Intended Audience and Reading Suggestions

- This document is intended for all stakeholders in the company and can be read by anyone with proper knowledge about the product. Developers and project managers should refer to the Overall Description and System Features section to familiarize themselves with the code and algorithms used in the product.

1.4 Product Scope

- MemTrain is a simplistic memory trainer game designed for the summer version of Comp 380 at California State University at Northridge. The objective was to design a fully functional game in a six week time span that proved basic understanding of the Android development environment as well as proficiency in Java and XML. This product was intended for non-commercial use and as such there is no intended business strategy being used.

1.5 References

Product Owner - Daniel Rojas

Scrum Master - Eric Hernandez

Development Team - Ade Bello , David Pineda, Eric Hernandez, Daniel Rojas

2. Overall Description

2.1 Product Perspective

- As stated in Section 1.4 of the SRS, this product was intended for non-commercial use and was made strictly for learning purposes. It is a self-sustaining product that does not rely on any other products for functionality. Please refer to the 'Assumptions and Dependencies' heading in this section for specific information on constraints and dependencies within the application itself. The UML diagram also attached to the SRS shows major components and functions of the entire system.

2.2 Product Functions

Please refer to Section 3 for a technical explanation of key system features and function. Alternatively, please view the diagrams for more information such as the object class diagram.

A brief summary of major application features include the following:

- Play Game - Start the main game
- Exit Game - Exit the application
- Help Menu - Open up help screen
- Return to Main Menu - Return back to the main menu from main game
- High Score Menu - Open up the leaderboard
- Press Button - Bitmap interaction with user
- Release Button - Bitmap interaction with user to release button and register index
- Sequence Randomization - Pattern randomization in main game
- Sequence Storage – Pattern storage in main game (both for system and user)
- Sequence Comparison – Comparison algorithm to compare user sequence to system sequence
- Timer – Simple UI Timer

2.3 User Classes and Characteristics

- public class MainActivity extends Activity : The initial starting class of the application used to enter into other more important activities.
- public class ButtonsMain extends View implements MemTrain.Listener
    - private void drawButtons – Positions the series of buttons on the canvas
    - private void drawButton : Positions a button on the canvas
    - getBitmapForButton : Decides which Bitmap image to display using switch cases.
    - getButtonIndex : Retrieves button position index
    - private int chooseDimension : Chooses dimensions of the canvas map
    - private Boolean onTouchEvent : Uses MotionEvent to handle interactions with the user and register button indexes.
    - public void buttonStateChanged : Calls invalidate to check for a single button change
    - public void multipleButtonStateChanged : Calls invalidate to check for multiple button state change

- public class Credits extends Activity: The simple class that controls the credits activities and its transitions between activities.
- public class Help extends Activity: The simple class that controls the help activity and its transitions between activities.
- public class HighScore extends Activity: The simple class that displays the user's overall highest score since their first time playing the game (unless reset by the user themselves).
- public final class MemTrain extends Activity : Creates the main methods and sound used for game functions
    - public interface Listener : Calls buttonStateChanged and multipleButtonStateChanged to invalidate buttons
    - public void showButtonPress : Play sound on specific gameMode type
    - public void releaseButton : Check's user interaction with system sequence and determines whether the game should progress or if the game is over.
    - public void showButtonRelease : Calls for sound and corresponding animation on release.
    - public void isButtonPressed: Checks for button press
    - public Bundle saveState : Saves all actives state of the game.
    - public void restoreState : Restores all active states of the game
    - public void gameStart : Start the game, begins the randomization
    - public void getRandomColor: : Choose a color to play in the playNext method
    - class UpdateHandler : Generic handler used for updates and delays
    - void scaleBeepDuration : Sets how long the beep should be upon randomization
    - public void update() : Set a delay for the animations being lit and check the gameMode and update mUpdateHandler
    - public void playNext : Randomize a sequence, store the sequence, show the appropriate animation, update the sequence indexes.
    - public MemTrain: Creates a model of MemTrain with active sounds
    - public void saveHighScore: Saves high score of the user
    - public int loadHighScore: Loads the user's highest score
    - public void playLast: utilized by the “Play Last” button to replay the latest sequence shown by the game

2.4 Operating Environment

- The application was built and designed to be used with the latest version of Android 4.0 and was optimized using the Nexus 5 API in Android Studio. The application will run on any Android device with these specifications.

2.5 Design and Implementation Constraints

- Hardware limitations include the inability to run the application on anything besides an Android device running Android 4.0. See ‘Assumptions and Dependencies’ for more specific limitations of the application.

2.6 User Documentation

- No physical documentation will be provided for the user. Within the main menu of the application there will be options providing virtual assistance similar to a tutorial to provide the user proper context and information about the game. Documentation linked to the developer credits will also be mapped out on the main menu for the user.

2.7 Assumptions and Dependencies

- This product will only work for Android devices running the latest version of Android 4.0. Functionality cannot be guaranteed for any devices running any other version. Alternatively, the program can be loaded in the Android Studio environment and launched with the HAXm Emulator. Using the emulator puts a constraint on the performance of the application and does not reflect the true performance on actual Android devices.
- The java files and XML files are also dependent on each other and the program cannot function properly if these files are changed or moved. The pictures located in the drawable folder of the application are also dependent on the java and XML files and cannot be renamed unless the name is also changed in the respective java and XML files.

3.External Interface Requirements

3.1 User Interfaces

- Main Menu - consisting of 3 buttons:
  - A play button that takes the user to the game window where the game itself is played.
  - A help & options button that takes the user to the help window which provides the user with a short tutorial as to how the game functions.
  - A high score button that takes the user to the high score screen where they may view their top score so far.
- Game window - consisting of high score counter and 6 buttons:
  - A start button that begins the game and shows the initial sequence to the user.
  - A play last sequence button that allows the user to playback the most recent sequence at the risk of losing score points.

    - 4 sequence buttons that are arranged in a 2 by 2 arrangement and are what display the sequence the user and take the user's sequence input.
- High score window - consists simply of the user's highest score at the top of the screen.
- Help window consists of a short tutorial for the user to fully understand how to play the game as well as a button that directs them to the credits window.
- Credits window consists of the names of the developers.

3.2 Hardware Interfaces

- The application is supported on android devices. The application receives and delivers sequences to and from the user via the screen of the device. The sequence that is generated for the user to mimic is shown on the screen. The user’s attempt to mimic the sequence is input via the touch screen of the device. All other communication between the user and the system is done through the device’s touch screen as well.

3.3 Software Interfaces

- The application must have interaction between the game window and highscore window where scores are kept track and stored if they meet the requirement of being within the highest three scores the user has achieved.
- The application must utilize image and sound files while the game is in progress and must project animations accordingly.

3.4 Communications Interfaces

- The application does not require any form of network communication. This includes email, web browser, electronic forms, etc. There is no use of FTP, HTTP, or anything similar in the application. Data transferring, encryption, and synchronization are not necessary for the application.

4.System Features

4.1 Play Game

4.1.1 Description and Priority

This feature is the main part of the system. It is the section where users are able to actually play the Memory Trainer game. Interactions with the system are done through user gestures such as pressing on the screen. This feature is of the highest priority. If Play Game does not function properly, then the Memory Trainer system is considered a failure.

4.1.2 Stimulus/Response Sequences

1. Users presses the start button, then system flashes a light sequence with buttons
2. User mimics sequence correctly, then system updates user game score and flashes a longer sequence
3. User mimics sequence incorrectly, then system terminates game process and resets users score
4. User presses play last button, then system repeats the most recent sequence and decreases score points.

4.1.3 Functional Requirements

REQ-1: The system shall generate a light sequence for the user to follow

REQ-2: The system shall keep track of user response pattern
REQ-3: The system shall keep track of user score and update the display accordingly
REQ-4: The system shall notify the user when a new high score has been reached
REQ-5: A user will be notified, via sound, when they have failed to match the sequence
REQ-7: The system shall restart the game whenever the user fails to match the sequence
REQ-8: A user shall be able to replay the most recent sequence and lose points
REQ-9: The system shall allow the user to exit the game and return to the main menu

4.2 Session

4.2.1 Description and Priority

This feature is essentially the main menu. It is where the user will be able to decide to play the Memory Trainer game view help options, view the high score, and exit the game. This feature is of high priority because if it does not function well, then the user will not be able do anything and the system would be useless.

4.2.2 Stimulus/Response Sequences

1. User presses start game button, then the system will begin the game
2. User presses the high score button, then the system will display the overall high score
3. User presses the help buttons, then the system will a tutorial for the game as well as the option to view the game credits.
4. User presses the back button, then the system quits the application

4.2.3 Functional Requirements

REQ-1: The system shall allow the user to navigate around the application

REQ-2: The system will allow the user to begin playing the Memory Trainer game

REQ-3: The user will be able to view the help and credits activities

REQ-4: The user will be able to view the overall high score of the game

REQ-5: The user shall be able to exit the application

4.3 High Score

4.3.1 Description and Priority

This feature allows the user to view the Memory Trainer game's overall high score. This feature has a medium priority. The play game functionality should come before the high score. Although, it is still very important for the user to be able to track their achievements.

4.3.2 Stimulus/Response Sequences

1. The user presses the high core button from session menu, then the system displays the overall high score.
2. The user presses the menu button, then the system reverts back to the main menu

4.3.3. Functional Requirements

REQ-1: The system shall display the current high scored achieved in the game

REQ-2: The user shall be able to go back to the main session menu via pressing the menu button

4.4 Help

4.4.1 Description and Priority

This feature gives the user a description as to how the game functions, how to gain points, and how they will lose points. Its also allows access to the games credits.

4.4.2 Stimulus/Response Sequences

1. The user presses the help button from the menu, then the system display the text tutorial to them.

4.4.3 Functional Requirements

REQ-1: The system shall display the game tutorial.

4.5 Exit

4.5.1 Description and Priority

This feature allows the user to exit the Memory Trainer application. This feature is of the least importance since a user can manually exit the game themselves via natural android phone mechanics.

4.5.2 Stimulus/Response Sequences

1. User presses the physical back button on the device, then the application exits itself

4.5.3 Functional Requirements

REQ-1: The user shall be allowed by the system to exit the application

5. Nonfunctional Requirements

5.1 Performance Requirements

- The time is takes for the a user to get from the main menu to another activity should be less than half a second. This is so that the user can have a seamless experience without any lag. Thus, user navigation is a critical priority.

- The user should never experience more than half a second delay between any action. This is to ensure a smooth UI experience.

- The error ratio should be 1 to 1000. Since this is a simple system, it should not be susceptible to failures. The system should be as accurate as possible and perform well consistently.

- The system should never, at any point, stall and exit itself. Therefore, the system should be well organized in that it doesn't use excessive memory or have functions that take too long to complete.

5.2 Security Requirements

- In case of a system failure, the system should not lose any important information such as the user's high score. If the user is not able to access their personal information, then the system will have failed to give a good user experience.

5.3 Software Quality Attributes

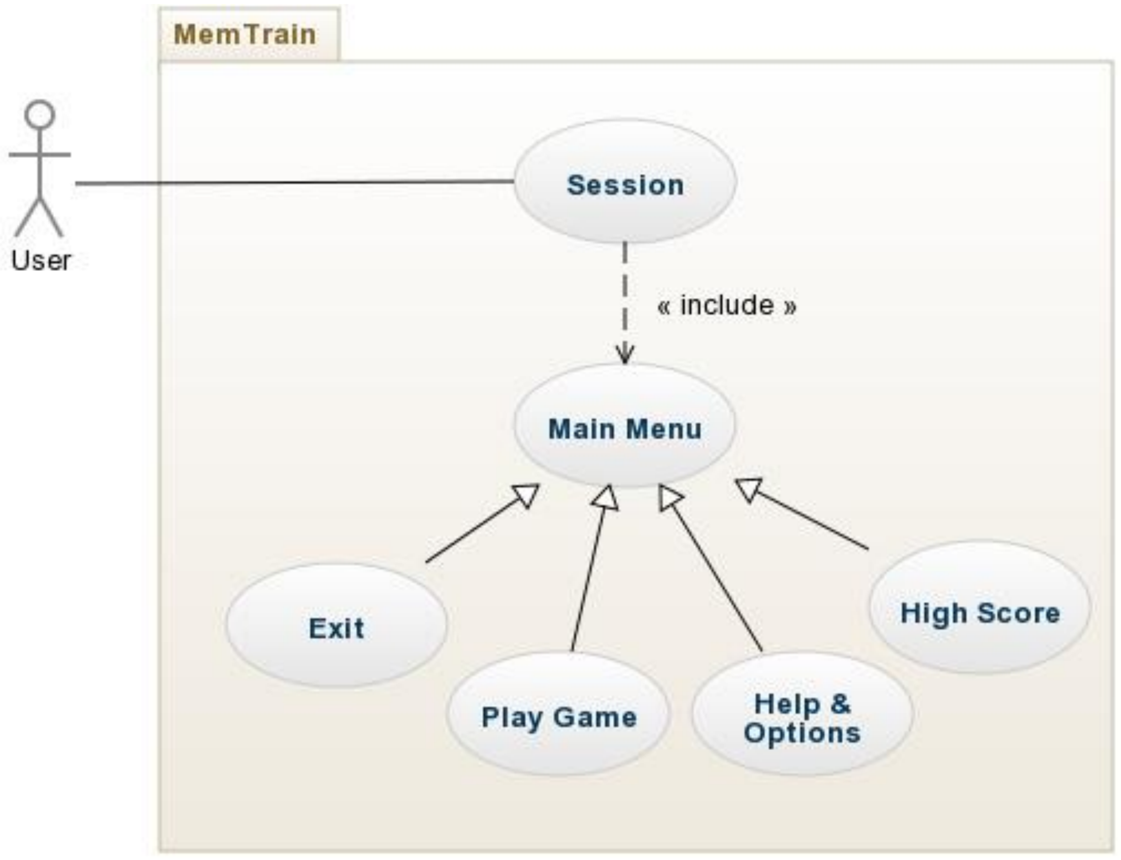
- The amount of training time it takes for the user to become familiar with the system software should be less than two minutes. This is because the system is a relatively simple game that should be easily understandable. If the user finds the system difficult to understand, then the system has failed.

6. Other Requirements

Appendix A: Glossary

- Emulator - hardware/software that allows a computer system to behave like another computer system
- Java - An object-oriented, general-purpose computer programming language based on classes
- UI - User Interface
- XML - Extensible Markup Language

Use Case Diagram



Use Case Descriptions

Title:	Session	
Description:	The system opens the main menu of the system. The user is able to navigate to any part of the system via button controls.	
Scenario:	Navigating system menu	
Triggering Events:	Open the MemTrain application	
Actors:	User	
Stakeholders:	User, developer, product owner, client	
Pre-condition:	User has device that is turned on	
Post-condition:	User is able to navigate the main menu	
Flow of Events:	Actor	System
	1. Press app icon on phone 3.Press button for desired option	2. Display Main Menu
Exception:	Step Condition	Action Description
	1a. App load up failure	Notify user and exit system

Title:	Play Game	
Description:	The system moves to the "Play Game" activity when the user selects the Start button from the main menu. User is able to play the MemTrain game.	
Scenario:	Playing the game	
Triggering Events:	Interacting with the start button on the main menu	
Actors:	User	
Stakeholders:	User, developer, product owner, client	
Pre-condition:	User has started the application and is in the main menu	
Post-condition:	System has entered the "Play Game" activity and user is able to interact with the systems main game.	
Flow of Events:	Actor	System
	1. Press the start button 3. Wait for system to start pattern 5. Follow pattern interaction until game is over.	2. Move to "Play Game" activity 4. System starts pattern/game
Exception:	Step Condition	Action Description
	2a.Move to "Play game" activity failure	2b. Notify user and exit system

Title:	Help & Options	
Description:	The system moves to the help activity when the user selects the help & options button from the main menu. The user is now given a text based tutorial about how to play the game. The user can also choose to view the game credits.	
Scenario:	Viewing game tutorial/credits	
Triggering Events:	User selects help & options button in the main menu of the system.	
Actors:	User	
Stakeholders:	User, developer, product owner, client	
Pre-condition:	User has started the application and is in the start up menu	
Post-condition:	User has viewed tutorial and is now knowledgeable enough to play the game. User also now knows the developers that created the game.	
Flow of Events:	Actor	System
	1. Press help & options button 4.Press credits button	2. Move to help activity 3. Display tutorial 5. Move to credits activity 6. Display game credits
Exception:	Step Condition	Action Description
	2a. Move to help activity failure 5a. Move to credits activity failure	Notify user and exit system Notify user and exit system

Title:	High Score	
Description:	The system opens the high score activity. Upon opening the high score activity, the screen displays the overall high score.	
Scenario:	Viewing of the previous high scores	
Triggering Events:	User selects the high score button in the main menu of the system.	
Actors:	User	
Stakeholders:	User, developer, product owner, client	
Pre-condition:	User has started the application and is in the startup menu	
Post-condition:	High score has been viewed by the user	
Flow of Events:	Actor	System
	1. Press high score button 3.Display the user's high score(s)	2. Move to high score activity
Exception:	Step Condition	Action Description
	2a. Move to high score activity failure 3a. No scores to display	Notify user and exit system Display "Score: 0"

Title:	Exit	
Description:	Termination of the system run process. It will end all activities and save useful data like high scores before fully terminating.	
Scenario:	Terminating application	
Triggering Events:	Pressing physical hardware exit buttons	
Actors:	User	
Stakeholders:	User, developer, product owner, client	
Pre-condition:	User has started the application and is in the startup menu	
Post-condition:	User has successfully exited the system	
Flow of Events:	Actor	System
	1. Press recent apps button 3. Shut down running system	2. Save all important information
Exception:	Step Condition	Action Description
	1a. System freezes	Notify user and exit system

CRC Cards

MainActivity	
Starts the main menu	MemTrain
Lets user access game	HIghScore
Lets user exit system	Help
Knows which buttons in menu have been pressed	

HighScore	
Knows stored user high score	MainActivity
Displays user high score	
Returns to main menu	

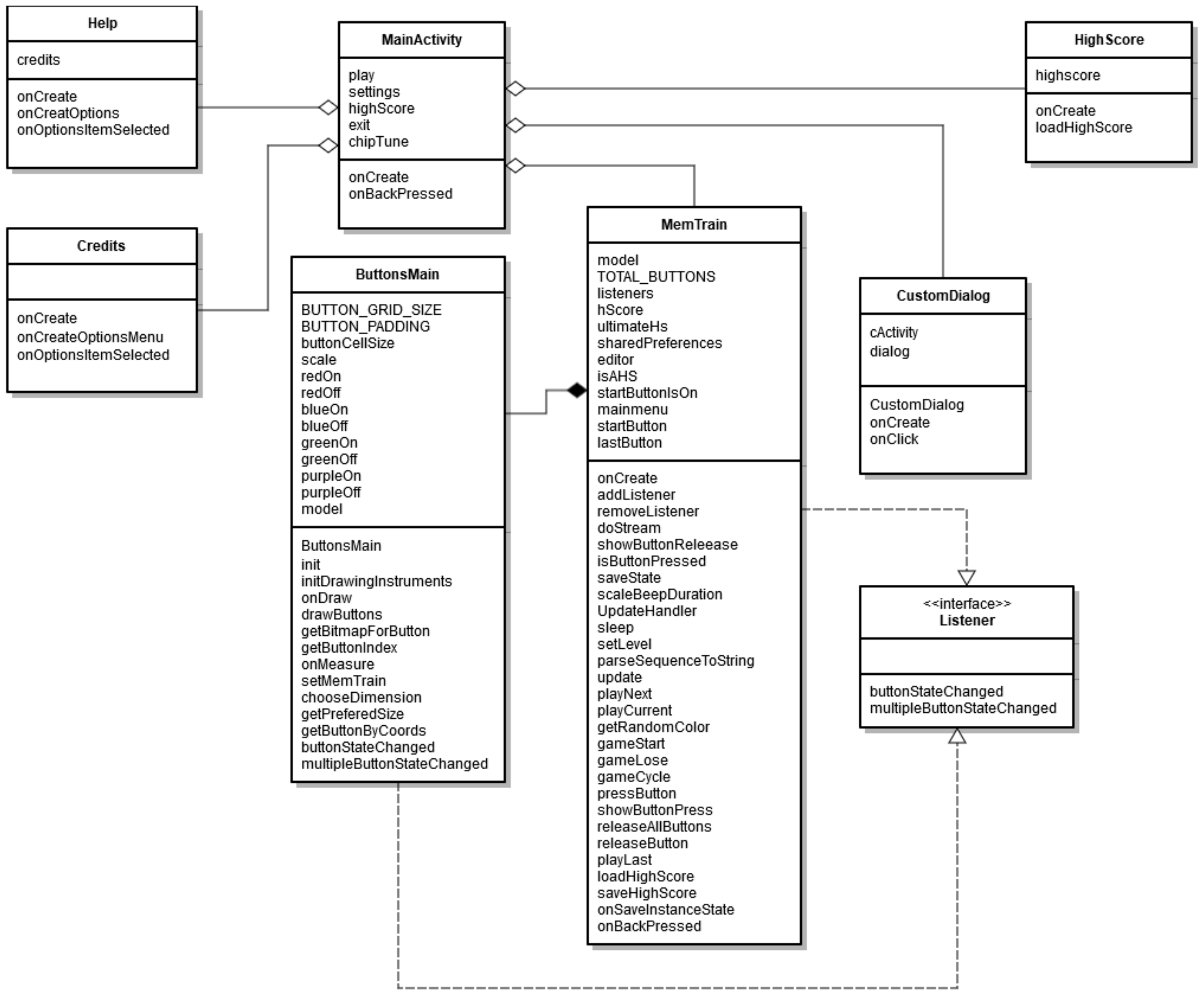
ButtonsMain	
Draws canvas for 4 buttons	MemTrain
Measures dimensions of all buttons	
Places all buttons on canvas	
Calls /drawable/ for picture buttons	
Calls /drawable for animation change pictures	
Sets up motion event to allow user interaction	

Help	
Tells user how to play game	MainActivity
Able to access game credits	Credits
Returns to main menu	

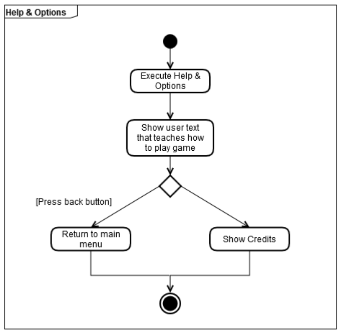
Credits	
Displays developers names	Help
Returns to the Help activity	

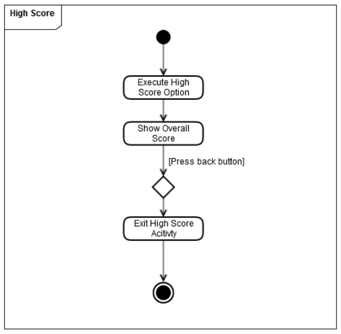
MemTrain	
Lets user play game	ButtonsMain
Sets up MemTrain object with soundpool for sound functionality	MainActivity
loads /raw/ sounds and binds them to specific buttons	
Plays random sequence withe corresponding button animation and sound	
Stores and increases complexity of sequence	
Measures user sequence	
Compares user sequence based on system sequence	
Knows and updates user high score	
Lets user know if new high score	
Lets user know of failure	
Lets user play same sequence, but deducts points	

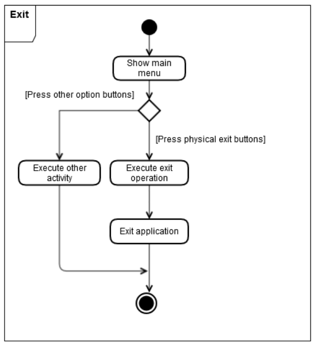
Class Diagrams

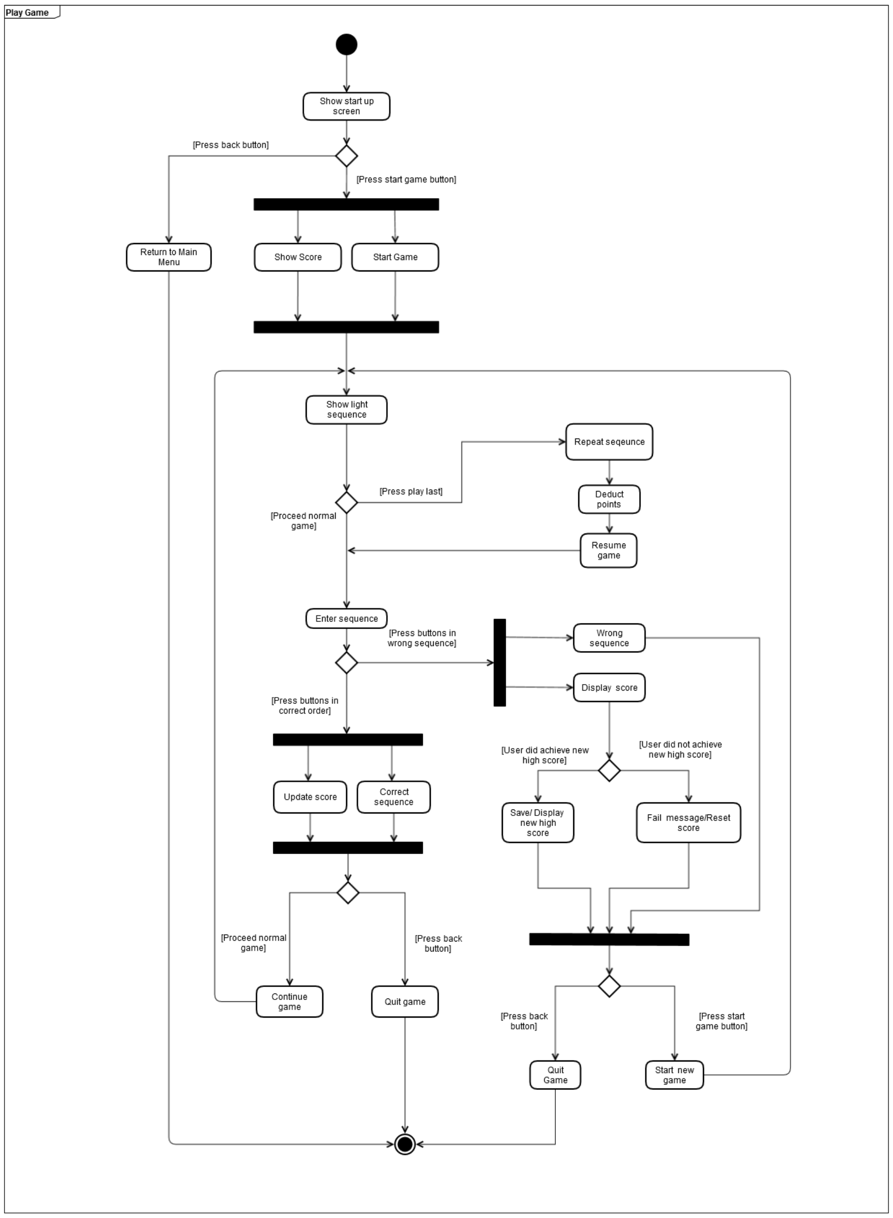


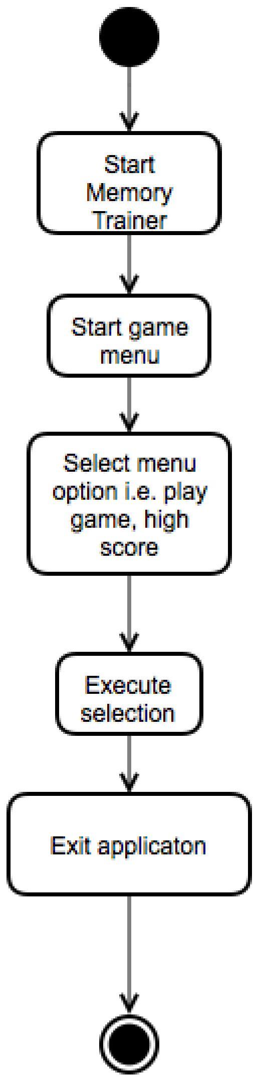
Activity Diagrams



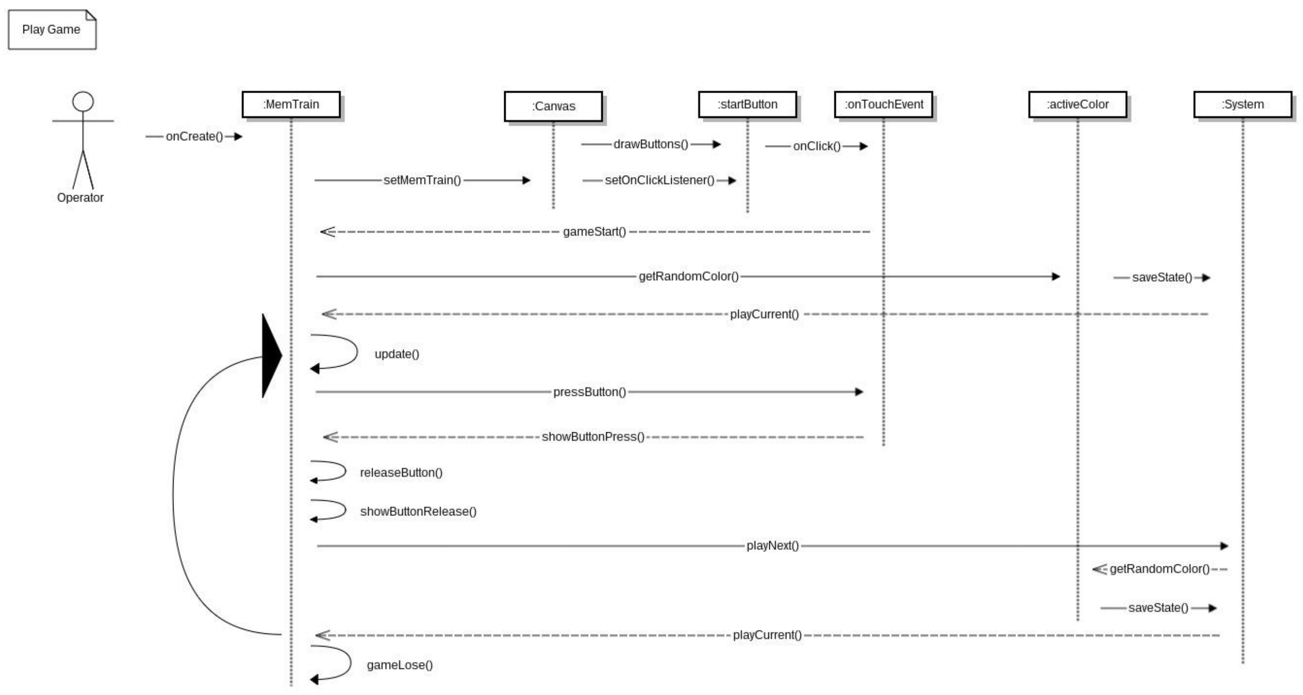


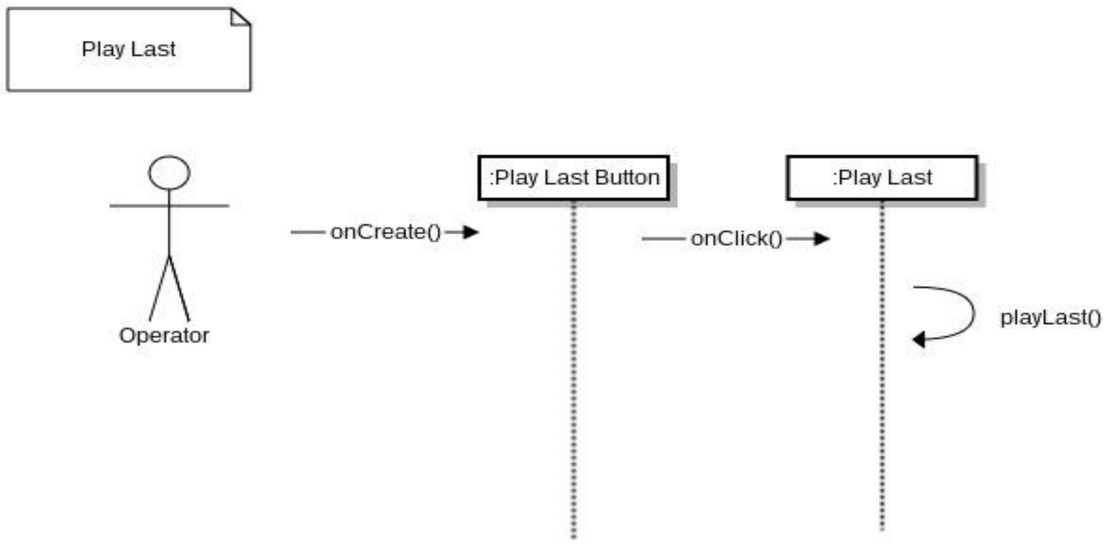


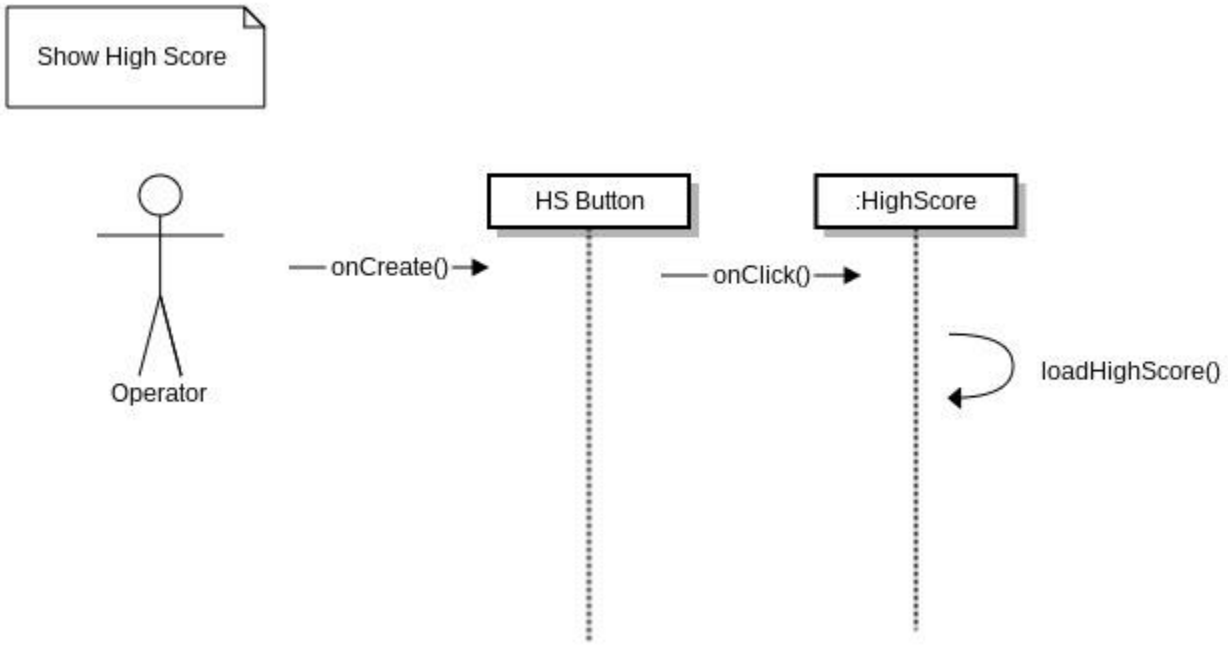


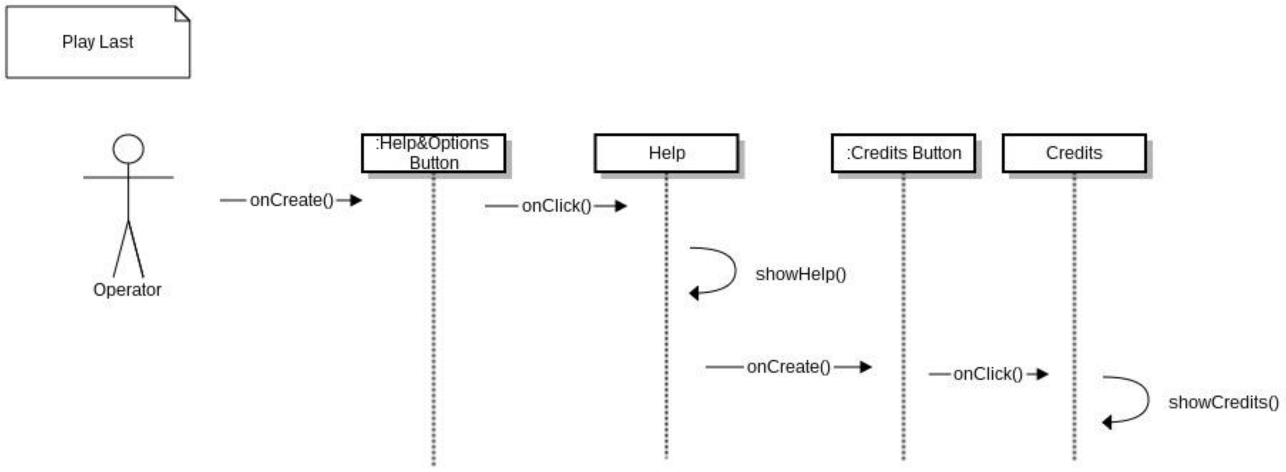

Start Memory Trainer
Start game menu
Select menu option i.e. play game, high score
Execute selection
Exit applicaton

Sequence Diagrams






Show High Score
Operator
HS Button
:HighScore
onCreate()
onClick()
loadHighScore()


Play Last
Operator
:Help&Options Button
Help
:Credits Button
Credits
onCreate()
onClick()
showHelp()
onCreate()
onClick()
showCredits()

1 TEST PLAN IDENTIFIER

Memory Trainer MTP 01.0

2 REFERENCES

- Memory Trainer SRS
- Use case diagram
- System-level activity diagram
- Function-level activity diagrams
- System-level sequence diagram
- Function-level sequence diagrams
- Class diagram
- CRC cards

3 INTRODUCTION

This document describes the master plan for testing the Memory Trainer system. The main objectives of this plan are to identify existing project information, identify the software components that should be tested, and describe the testing strategies to be employed. This plan will also reference other resources such as the documents described in the previous section, which will aid in the testing procedure. The types of testing that will be performed include unit testing, product testing, and white box testing.

4 TEST ITEMS (FUNCTIONS)

- Memory Trainer version 1.0
- System use-cases
- Memory Trainer class files
- Memory Trainer xml files

5 SOFTWARE RISK ISSUES

- Methods need to be handled with extreme care
- Null references
- Fatal errors with new activities
- Redundant code
- Excessive memory usage
- Performance degradation

6 FEATURES TO BE TESTED

- User’s ability to open the application [H]
- User’s ability to begin game by pressing “play game” button [H]

- User’s ability to view given sequence and input attempted mimicking of the sequence [H]
- User’s ability to see sequence animation without any animation/sound issues [H]
- User’s ability to easily navigate application through button presses [H]
- User’s ability to view their score as it increases with in-game progress [M]
- User’s ability to replay sequence as they wish [M]
- User’s ability to exit game without any issues [M]
- User’s ability to view highscore while not in-game [L]
- User’s ability to view tutorial and credits by pressing the “tutorial” button in the main menu [L]

7 FEATURES NOT TO BE TESTED

Given the scope of the project, the size of the system allows for all features to be tested to an acceptable level of performance.

8 APPROACH (STRATEGY)

Testing will be done through the Android Studio virtual device and on an Android phone.No training is required for the use of the Android phone for testing. Using the Android Studio virtual device does not require training as well. Time measurements will be collected when testing time of application startup, time between sequence animation & sound, and time of application shutdown. As the application is developed, added features are tested to determine issues which need to be resolved. Regression testing will be done as the application develops. Each bug will be addressed and resolved as it is found. Applications features will be tested once thought to have been completed for bugs and any other improvement. In-game animation and sound of the sequence display animation must be tested together for optimal performance.

9 ITEM PASS/FAIL CRITERIA

- PASSING
    - Startup of the application must be seamless with no notable delay to the user
    - Button input should always cause an appropriate system response
    - Game animation & sounds should have minimal delay in between them and should coordinate together perfectly.
    - The system should never exit without user’s intention to exit
    - The system should not use excessive amount of memory
- FAILING
    - Startup time is not quick
    - Buttons are irresponsive
    - Game animation & sounds do not coordinate
    - System requires large amounts of memory

10 SUSPENSION CRITERIA AND RESUMPTION REQUIREMENTS

If a defect in any feature is found to be too difficult to amend, the feature will be developed using a different approach and scrapped if found to be far too difficult to fix.

11 TEST DELIVERABLES

Test Cases, Test Logs, Proof of functionality from HAXm Emulator, Error Logs, Execution Logs, Manifest Files.

12 REMAINING TEST TASKS

"Play Game" Activity Stress Test
"High Score" Activity Functionality Test
Android Device Full Application Test

13 ENVIRONMENTAL NEED

HAXm Android Environment Simulator
Physical device running Android 4.0.3 Ice Cream Sandwich

14 STAFFING AND TRAINING NEEDS

No training or extra staffing is necessary. All test cases are run directly from Android Studio and require no extra resources.

15 RESPONSIBILITIES

Product Owner – Daniel Rojas

Makes final changes to the product according to the outcome of the test cases

Scrum Master – Eric Hernandez

Makes final changes to the business plan and schedule according to outcome of test cases.

Development Team – Ade Bello, David Pineda

Responsible for designing and executing test cases and judging runtime performance and optimization.

16 SCHEDULE

Time has been allocated within the project plan for the following testing activities. The specific dates and times for each activity might vary due to small time constraints. Coordination of the personnel required for each task, test team, development team and management will be handled by the project manager with the help of test leaders.

A. Review of Requirements document by team members and initial creation of project classes, subclasses and objectives to be met in the duration of the project.

B. Development of Master test plan by test manager and test with time allocated for at least two weeks.

C. Review of the design document by the entire test team. This will provide the test team with a clear understanding of the the way the application will be structured and will further define the main classes, subclasses and objectives.

D. Development of acceptance test plans by test manager and test teams with time allocated for reviews.

E. Review of the detail design document by test team personnel. This will provide the team with a clearer understanding of the individual program structure giving them a better idea of what to do.

F. Time set aside to implement changes made to the test process.

17 PLANNING RISKS AND CONTINGENCIES

A. Limited time for project completion.

Currently, due to the course being only 6 weeks long, the entire team is under a very tight time constraint. Due to such situation, the team might delay on reviewing appropriate documents and the entire team might not participate in the acceptance test process. If the time constraint should become a problem, the dates for reviews and acceptance testing will be handled by few members of the team while the rest of the team handles other time sensitive things so the project does not become delayed.

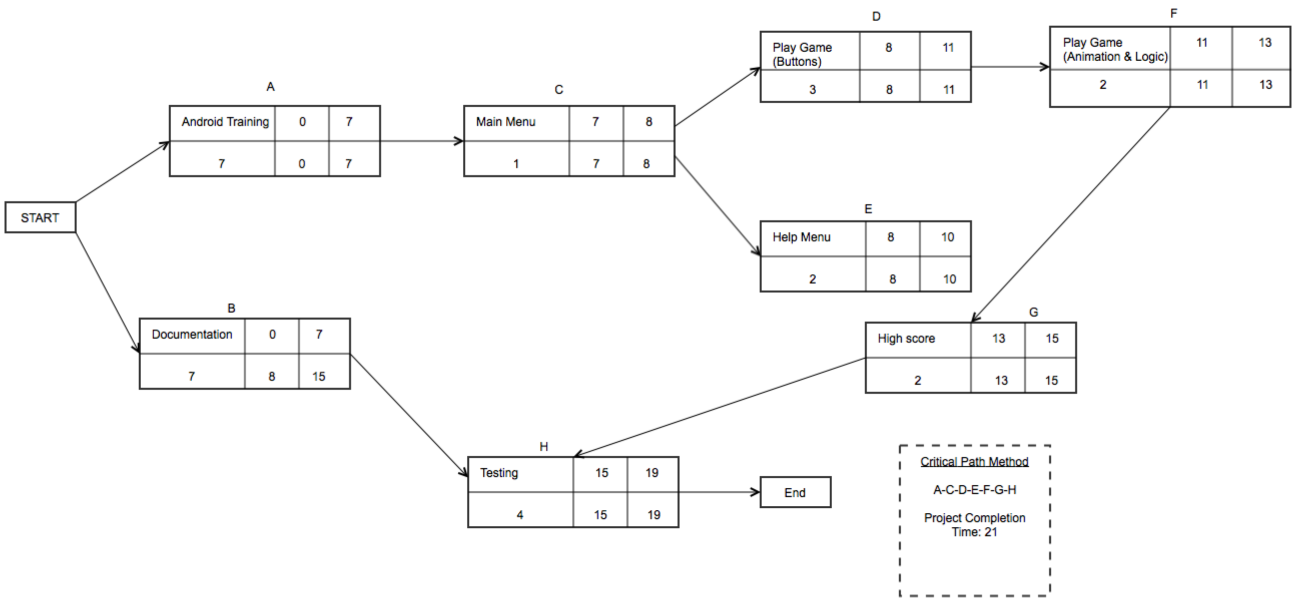
18 APPROVALS

Project Sponsor - Dr. George Wang	
Development Management - Eric Hernandez	
Project Manager - Eric Hernandez	
Test Manager - Ade Bello	
Development Team Manager - Ade Bello	

19 GLOSSARY

CRC Cards - Class-responsibility-collaboration cards
HAXm - a virtualization engine that utilizes Intel Virtualization Technology for emulation of Android devices
SRS - Software Requirements Specification

Critical Path



MainActivity.java

```
package daniel.memtrain;

import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.app.Activity;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;


public class MainActivity extends Activity {
    MediaPlayer chipTune; //our menu music
    ImageButton play, help, highScore, exit; //our menu buttons

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final Intent newGame = new Intent(this, MemTrain.class);
        final Intent hsPage = new Intent(this, HighScore.class);
        final Intent helpPage = new Intent(this, Help.class);

        chipTune = MediaPlayer.create(this, R.raw.jakim); //create music
        chipTune.setLooping(true); //will loop infinitely
        chipTune.start(); //start immediately when game opens

        //initialing some buttons for the main activity
        play = (ImageButton) findViewById(R.id.startGame);
        play.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()) {
                    case MotionEvent.ACTION_DOWN:
                        play.setImageResource(R.drawable.sgametwo);
                        break;
                    case MotionEvent.ACTION_UP:
                        play.setImageResource(R.drawable.sgameone);
                        chipTune.pause();
                        startActivity(newGame);
                        break;
                }
                return false;
            }
        });

        highScore = (ImageButton) findViewById(R.id.highScore);
        highScore.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()) {
                    case MotionEvent.ACTION_DOWN:
                        highScore.setImageResource(R.drawable.highscoretwo);
                        break;
                    case MotionEvent.ACTION_UP:
                        highScore.setImageResource(R.drawable.highscoreone);
                        startActivity(hsPage);
                        break;
                }
                return false;
            }
        });

        help = (ImageButton) findViewById(R.id.help);
        help.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()) {
                    case MotionEvent.ACTION_DOWN:
                        help.setImageResource(R.drawable.helptwo);
                        break;
                    case MotionEvent.ACTION_UP:
                        help.setImageResource(R.drawable.helpone);
                        startActivity(helpPage);
                        break;
                }
                return false;
            }
        });
    }
}
```

ButtonsMain.java

```
package daniel.memtrain;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.util.FloatMath;
import android.view.MotionEvent;
import android.view.View;

public class ButtonsMain extends View implements MemTrain.Listener {

	// measurements
	private static final int BUTTON_GRID_SIZE = 2;
	private static final float BUTTON_PADDING = 0.01f;
	private float buttonCellSize;
	private float scale;

	// drawing tools. we use bitmap to manage our animation states
	private Bitmap redOn;
	private Bitmap redOff;
	private Bitmap blueOn;
	private Bitmap blueOff;
	private Bitmap greenOn;
	private Bitmap greenOff;
	private Bitmap purpleOn;
	private Bitmap purpleOff;
	private Bitmap buttonPressed;
	private Bitmap buttonIdle;

	// model
	private MemTrain model;

	public ButtonsMain(Context context, AttributeSet attrs, int defStyle) {
		super(context, attrs, defStyle);
		init();
	}

	public ButtonsMain(Context context, AttributeSet attrs) {
		super(context, attrs);
		init();
	}

	public ButtonsMain(Context context) {
		super(context);
		init();
	}

	private void init() {
		initDrawingInstruments();
	}

	private void initDrawingInstruments() { //here we are assigning our pictures to our different states
		Resources resources = getContext().getResources();
		redOn = BitmapFactory.decodeResource(resources, R.drawable.redshape_on);
		redOff = BitmapFactory.decodeResource(resources, R.drawable.redshape_off);
		blueOn = BitmapFactory.decodeResource(resources, R.drawable.blueshape_on);
		blueOff = BitmapFactory.decodeResource(resources, R.drawable.blueshape_off);
		greenOn = BitmapFactory.decodeResource(resources, R.drawable.greenshape_on);
		greenOff = BitmapFactory.decodeResource(resources, R.drawable.greenshape_off);
		purpleOn = BitmapFactory.decodeResource(resources, R.drawable.purpleshape_on);
		purpleOff = BitmapFactory.decodeResource(resources, R.drawable.purpleshape_off);
	}

	@Override
	protected void onDraw(Canvas canvas) {
		super.onDraw(canvas);
		scale = canvas.getClipBounds().width();
		canvas.save(Canvas.MATRIX_SAVE_FLAG);
		canvas.scale(scale, scale);
		drawButtons(canvas);
		canvas.restore();
	}

	private void drawButtons(Canvas canvas) {
		for (int row = 0; row < BUTTON_GRID_SIZE; ++row) {
			for (int col = 0; col < BUTTON_GRID_SIZE; ++col) {
				drawButton(canvas, row, col);
			}
		}
	}

	private void drawButton(Canvas canvas, int row, int col) {
		buttonCellSize = 1.0f / BUTTON_GRID_SIZE;

		float buttonCellTop = row * buttonCellSize;
		float buttonCellLeft = col * buttonCellSize;
		float buttonTop = buttonCellTop + BUTTON_PADDING;
		float buttonLeft = buttonCellLeft + BUTTON_PADDING;
		float buttonSize = (buttonCellSize - BUTTON_PADDING * 2);

		Bitmap bitmap = getBitmapForButton(row, col);

		float pixelSize = canvas.getClipBounds().width();
		float bitmapScaleX = (pixelSize / bitmap.getWidth()) * buttonSize;
		float bitmapScaleY = (pixelSize / bitmap.getHeight()) * buttonSize;

		canvas.save(Canvas.MATRIX_SAVE_FLAG);
		canvas.scale(bitmapScaleX, bitmapScaleY);
		canvas.drawBitmap(bitmap, buttonLeft / bitmapScaleX, buttonTop / bitmapScaleY, null);
		canvas.restore();
	}
```

```
private Bitmap getBitmapForButton(int row, int col) {
        boolean pressed = model.isButtonPressed(getButtonIndex(row, col));
        int button_number = getButtonIndex(row, col);
        switch (button_number) {
        case 0:
                return pressed ?
                                greenOn : greenOff;
        case 1:
                return pressed ?
                                redOn : redOff;
        case 2:
                return pressed ?
                                purpleOn : purpleOff;
        case 3:
                return pressed ?
                                blueOn : blueOff;
        default:
                return pressed ?
                                buttonPressed : buttonIdle;
        }
}

private int getButtonIndex(int row, int col) {
        return row * BUTTON_GRID_SIZE + col;
}

@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int widthMode = MeasureSpec.getMode(widthMeasureSpec);
        int widthSize = MeasureSpec.getSize(widthMeasureSpec);

        int heightMode = MeasureSpec.getMode(heightMeasureSpec);
        int heightSize = MeasureSpec.getSize(heightMeasureSpec);

        int chosenWidth = chooseDimension(widthMode, widthSize);
        int chosenHeight = chooseDimension(heightMode, heightSize);

        int chosenDimension = Math.min(chosenWidth, chosenHeight);
        setMeasuredDimension(chosenDimension, chosenDimension);
}

public void setMemTrain(MemTrain model) {
        if (model != null) {
                model.removeListener(this);
        }
        this.model = model;
        if (model != null) {
                model.addListener(this);
        }
```

```
		}
		private int chooseDimension(int mode, int size) {
			if (mode == MeasureSpec.AT_MOST || mode == MeasureSpec.EXACTLY) {
				return size;
			} else { // (mode == MeasureSpec.UNSPECIFIED)
				return 300;
			}
		}

		@Override
		public boolean onTouchEvent(MotionEvent event) {
			int buttonIndex = -1;
			switch (event.getAction()) {
			case MotionEvent.ACTION_DOWN:
				buttonIndex = getButtonByCoords(event.getX(), event.getY());
				if (buttonIndex != -1) {
					model.pressButton(buttonIndex);
				}
				return true;
			case MotionEvent.ACTION_UP:
				buttonIndex = getButtonByCoords(event.getX(), event.getY());
				if (buttonIndex != -1) {
					model.releaseButton(buttonIndex);
				}
				model.releaseAllButtons();
				return true;
			}
			return false;
		}

		private int getButtonByCoords(float x, float y) {
			float scaledX = x / scale;
			float scaledY = y / scale;

			float buttonCellX = FloatMath.floor(scaledX / buttonCellSize);
			float buttonCellY = FloatMath.floor(scaledY / buttonCellSize);

			return getButtonIndex((int) buttonCellY, (int) buttonCellX);
		}

		//invalidate tells our system to redraw using onDraw upon a view change
		public void buttonStateChanged(int index) { invalidate();}

		public void multipleButtonStateChanged() {
			invalidate();
		}
	}
```

MemTrain.java

```
package daniel.memtrain;

import java.util.*;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Typeface;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.content.Context;

import android.media.AudioManager;
import android.media.SoundPool;
import android.support.v7.app.AppCompatActivity;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.TextView;
import android.widget.Toast;

/**
 * Created by daniel on 7/13/15.
 */
public final class MemTrain extends Activity{

    private MemTrain model; //new MemTrain object
    public static final int TOTAL_BUTTONS = 3 * 3; //9 buttons superfluous, only 5 in use
    private List<Listener> listeners = new ArrayList<Listener>();
    private static TextView e;
    private static int hScore = 0, ultimateHs;//HighScore
    private static SharedPreferences sharedPreferences;
    private static SharedPreferences.Editor editor;
    private static boolean isAHS = false, startButtonIsOn = false;;
    private static Intent mainmenu;
    private static ImageButton startButton;
    private static ImageButton lastButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        model = new MemTrain(this); //new object
```

```
//creating font and view
super.onCreate(savedInstanceState);
setContentView(R.layout.game_activity_layout);
e = (TextView)findViewById(R.id.eric_hs);
String fontPath = "font/arcadepix.TTF";
Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
e.setTypeface(tf);
e.setText("Score: " + hScore);

mainmenu = new Intent(this, MainActivity.class);
sharedPreferences = getSharedPreferences("MyData", Context.MODE_PRIVATE);
editor = sharedPreferences.edit();
ultimateHs = loadHighScore();

ButtonsMain grid = (ButtonsMain) this.findViewById(R.id.button_grid);
grid.setMemTrain(model); //new interactable button grid set up for our memtrain model

startButton = (ImageButton)findViewById(R.id.start); //start button
startButton.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                startButton.setImageResource(R.drawable.playgamestwo);
                break;
            case MotionEvent.ACTION_UP:
                startButton.setImageResource(R.drawable.playgamesone);
                mUpdateHandler.postDelayed(new Runnable() {
                    public void run() {
                        startButtonIsOn = true;//ADDED
                        hScore = 0;
                        e.setText("Score: " + hScore);
                        model.gameStart();
                    }
                }, 100);
                break;
        }
        return false;
    }
});

lastButton = (ImageButton) findViewById(R.id.last_button);
lastButton.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch(event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                lastButton.setImageResource(R.drawable.playgameltwo);
```

```
                    break;
                case MotionEvent.ACTION_UP:
                    lastButton.setImageResource(R.drawable.playgamelone);
                    if(gameMode != LOST) {
                        model.playLast();
                    }
                    if (!startButtonIsOn) {

                    }
                    else {
                        if (hScore == 0) {
                            hScore = -10;
                        }
                        else if (hScore > 0){
                            if (hScore == 10) {
                                hScore = 0;
                            }
                            else {
                                hScore = hScore / 2;
                            }
                        }
                        else {
                            hScore = hScore * 2;
                        }
                        e.setText("Score: " + hScore);
                    }
                    break;
            }
            return false;
        }
    });

    /* After all initialization, we set up our save/restore InstanceState Bundle. */
    if (savedInstanceState == null) {                    // Just launched.  Set initial state.
        SharedPreferences settings = getPreferences (0); // Private mode by default.
        model.setLevel(settings.getInt(MemTrain.GAME_LEVEL, 1)); //setting level to default
    }
}

public interface Listener { //Listener uses invalidate button so pixels are redrawn in main thread
    void buttonStateChanged(int index); //calls invalidate()
    void multipleButtonStateChanged(); //calls invalidate()
}

public void addListener(Listener listener) {
    listeners.add(listener);
}
public void removeListener(Listener listener) {
    listeners.remove(listener);
```

```
}

private boolean[] buttonPressMap = new boolean[TOTAL_BUTTONS]; //boolean map of our buttons
private SoundPool soundPool; //soundPool mapped to animations
private int[] soundIds = new int[TOTAL_BUTTONS + 1];  // Add lose
private int speakerStream;

//doStream is similar to Mediaplayer's .start()
public void doStream(int soundId) {
    if (soundId != 0) {  // Don't do anything different if our soundID is invalid.
        if (speakerStream != 0) {  // Stop what we were doing.
            soundPool.stop(speakerStream);
        }
        speakerStream = soundPool.play(soundId, 1.0f, 1.0f, 0, 0, 1.0f);
    }
}

public void showButtonRelease(int index) { //play sound with button animation
    if (index >= 0 && index < TOTAL_BUTTONS) { //if our index exists
        if (buttonPressMap[index] == true) { //if our index in our buttonmap is on
            buttonPressMap[index] = false; //turn it off
            if (speakerStream != 0) {
                soundPool.stop(speakerStream); //stop speaker
                speakerStream = 0;
            }
            for (Listener listener : listeners) {
                listener.buttonStateChanged(index); //release button
            }
        }
    }
}

public boolean isButtonPressed(int index) { //check if individual button was pressed by index
    if (index < 0 || index > TOTAL_BUTTONS) { //if our index does not exist
        return false; //our button was not pressed
    } else {
        return buttonPressMap[index]; //return booelan result of the index
    }
}

//TIMERS
private static final int TICK_DURATION = 100;
private static final int BETWEEN_DURATION = 50;
private static final int TICK_COMPENSATION = TICK_DURATION - BETWEEN_DURATION;

//handler messages
private static final int UI = 0;

//game states
```

```
private static final int IDLE = 0;
private static final int LISTENING = 1;
private static final int PLAYING = 2;
private static final int LOSING = 8;
private static final int LOST = 9;

//sound management
private static final int GREEN = 0;
private static final int RED = 1;
private static final int YELLOW = 2;
private static final int BLUE = 3;
private static final int VICTORY_SOUND = 4; //currently used only for new high score
private static final int LOSE_SOUND = 5; //FAIL sound

//keys for game-state control
public static final String THE_GAME = "theGame";
public static final String GAME_LEVEL = "gameLevel";
private static final String SEQUENCE_INDEX = "sequenceIndex";
private static final String TOTAL_LENGTH = "totalLength";
private static final String PLAYER_POSITION = "playerPosition";
private static final String GAME_MODE = "gameMode";
private static final String WIN_TONE_INDEX = "winToneIndex";
private static final String IS_LIT = "isLit";
private static final String BEEP_DURATION = "beepDuration";
private static final String HEARD_BUTTON_PRESS = "heardButtonPress";
private static final String PAUSE_DURATION = "pauseDuration";
private static final String ACTIVE_COLORS = "activeColors";
private static final String CURRENT_SEQUENCE = "currentSequence";

private boolean[] activeColors = new boolean[4]; //array of the active colors on our board
private int[] currentSequence = new int[32]; //we keep track of current

private int sequenceLength;
private int sequenceIndex;
private int totalLength;
private int playerPosition;
private long beepDuration;
private long mLastUpdate;
private int gameMode;
private int winToneIndex;
private int theGame;

private static final Random RNG = new Random();
private boolean isLit;
private boolean heardButtonPress;  // Avoid a race of: down -> listen -> up.
private long pauseDuration;

public MemTrain(Context context) { //object
   for (int i = 0; i < TOTAL_BUTTONS; ++i) { //while there are no buttons in play
```

```
        buttonPressMap[i] = false; //no buttons are being pressed
    }

    //mapping soundIds to our button layout using SoundPool
    soundPool = new SoundPool(TOTAL_BUTTONS, AudioManager.STREAM_MUSIC, 0);
    soundIds[GREEN] = soundPool.load(context, R.raw.green, 1);
    soundIds[RED] = soundPool.load(context, R.raw.red, 1);
    soundIds[YELLOW] = soundPool.load(context, R.raw.purple, 1);
    soundIds[BLUE] = soundPool.load(context, R.raw.blue, 1);
    soundIds[VICTORY_SOUND] = soundPool.load(context, R.raw.mbison, 1);//Might need to fix
    soundIds[LOSE_SOUND] = soundPool.load(context, R.raw.lose, 1);

                    /* initializing */
    mLastUpdate = System.currentTimeMillis(); //generic update
    gameMode = IDLE; //game now waits for instruction
    isLit = false; //no buttons in play
    heardButtonPress = false; //no press allowed until gamestart
    winToneIndex = 0;

}

//save the state and various indexes of the game
public Bundle saveState(Bundle map) {
    if (map != null) {
        map.putInt(THE_GAME, Integer.valueOf(theGame));
        map.putString(CURRENT_SEQUENCE, parseSequenceToString(currentSequence, sequenceLength));
        map.putInt(SEQUENCE_INDEX, Integer.valueOf(sequenceIndex));
        map.putInt(TOTAL_LENGTH, Integer.valueOf(totalLength));
        map.putInt(PLAYER_POSITION, Integer.valueOf(playerPosition));
        map.putInt(WIN_TONE_INDEX, Integer.valueOf(winToneIndex));
        map.putLong(BEEP_DURATION, Long.valueOf(beepDuration));
        map.putLong(PAUSE_DURATION, Long.valueOf(pauseDuration));
        map.putBooleanArray(ACTIVE_COLORS, activeColors);
        map.putBoolean(IS_LIT, Boolean.valueOf(isLit));
        map.putBoolean(HEARD_BUTTON_PRESS, Boolean.valueOf(heardButtonPress));
        map.putInt(GAME_MODE, Integer.valueOf(gameMode));
    }
    return map;
}

void scaleBeepDuration(int index) { //beep length
    if (index < 6) beepDuration = 420;        //.42s
    else if (index < 14) beepDuration = 320; // .32s
    else beepDuration = 220;             //  .22s
    beepDuration -= TICK_COMPENSATION;
    if (beepDuration < 0) beepDuration = 0;
}

private UpdateHandler mUpdateHandler = new UpdateHandler(); //handler for delays/UI
```

```
class UpdateHandler extends Handler {

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case UI:
                MemTrain.this.update();
                break;
        }
    }

    public void sleep(long delayMillis) {
        this.removeMessages(UI);
        sendMessageDelayed(obtainMessage(UI), delayMillis);
    }
}

public void setLevel(int level) { //only one level for now, can set more with use cases
    int savedTotalLength = totalLength;

    switch (level) {
        case 1:
            totalLength = 32; //SET TO RUN A MAX PATTERN OF 32 WITHOUT LEVELS
            break;
        default:
            totalLength = 4;    // Should never be here but we need a default so whatever.
            break;
    }
    if (totalLength != savedTotalLength) {    // If we changed the game level we reset the game.
        if (pauseDuration > 0) pauseDuration = 0;  // Go directly to idle, and don't pause.
        if (isLit) playNext();      // If there's a button lit, turn it off.
        gameMode = IDLE;
        sequenceIndex = 0;
        // Can set sequenceLength to 0 but it works without it so yolo
    }
}

private String parseSequenceToString(int[] array, int length) { //generic string parser
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < length; i++) {
        sb.append(array[i]);
    }
    return sb.toString();
}

public void update() {
    long now = System.currentTimeMillis(); //update time to current
    long delay = beepDuration;    // Events are normally the length of a beep.
    if (isLit) delay = BETWEEN_DURATION;   //delay 50ms after turning off a lit light.

    if (gameMode != LISTENING) { //if we are not interacting with user
        if (now - mLastUpdate > delay) { //check if we updated already, if not
            playNext(); //head to playNext method
            mLastUpdate = now; //tell mLastUpdate we updated the sequence at this time
        }
        mUpdateHandler.sleep(delay);
    }
}

/*Main logic of the program to play next sequence with proper sound and animation as well as
 proper state updates */

public void playNext() {
    if (pauseDuration > 0) {  // delay check
        pauseDuration = 0;
        return;
    }
    switch (gameMode) {
        case PLAYING:  //  Play the current sequence if we are playing the game.
            if (sequenceIndex < sequenceLength) {    // Keep playing
                if (isLit) { //if our light is on
                    showButtonRelease(currentSequence[sequenceIndex]); // Stop previous tone.
                    isLit = false; //turn it off
                    sequenceIndex++; // Point at next index
                    if (sequenceIndex == sequenceLength) { // check to see if we played last tone
                        if (gameMode == PLAYING) {   // if we're playing begin listening for input.
                            sequenceIndex = 0;     // use sequenceIndex as match cursor.
                            gameMode = LISTENING;        // switch to Listen when button release feedback is done.
                        } else
                            gameMode = IDLE;  // or go to Idle state after replay.
                    }
                } else {
                    showButtonPress(currentSequence[sequenceIndex]);    // Flash and beep current.
                    isLit = true; //our light is now on
                }
            } // Fall through and do nothing if we're past the end of the sequence.
            break;
    }
}

public void playCurrent() { //update that system that we are playing currently
    gameMode = PLAYING;
    sequenceIndex = 0;
    update();
}

public int getRandomColor() { //RNG pick a color to beep
```

```
    int returnval = RNG.nextInt(4);
    if (theGame == 3) {            // Filter out inactive colors
        while (activeColors[returnval] == false)
            returnval = RNG.nextInt(4);  // Keep trying till we get an active.
    }
    return returnval;
}

public void gameStart() { //initiates MemTrain activity
    for (int i = 0; i < 4; i++) { //loop through all button indexes
        activeColors[i] = true;            // Mark all colors active.
    }

                    /* In case user was fast on the draw: Reset all buttons. */
    for (int index = 0; index < 4; index++) {
        showButtonRelease(index);
    }

    winToneIndex = 0;
    sequenceLength = 1;
    scaleBeepDuration(1);
    playerPosition = 1;
    currentSequence[0] = getRandomColor(); //value of our rng color determines current sequence index in our
array
    playCurrent(); //we are now playing the game
}

public void gameLose() {
    if (isAHS) {
        e.setText("New High Score!\n" + hScore);
        e.setGravity(Gravity.CENTER);
        isAHS = false;
        doStream(soundIds[VICTORY_SOUND]);
    }
    else {
        e.setText("FAIL!");
        doStream(soundIds[LOSE_SOUND]);
    }
    hScore = 0;
    startButtonIsOn = false;

    if (theGame == 3) {   // In game 3 we eliminate a color and start again.
        int activeColorCount = 0;
        for (int i = 0; i < 4; i++) {
            if (activeColors[i]) activeColorCount++;
        }
    } else {
        gameMode = LOST;
        update();
    }
}

public void gameCycle() {
    mLastUpdate = System.currentTimeMillis();
    pauseDuration = 800;        // Wait .8s after last key pressed to play next for game 1 and 3.
    playerPosition = 1;
    update();
    playCurrent();
}

        /* pressButton is called by the Touch Handler in response to the user... we calculator if the user is
         correct first, then we show the release animation */

public void pressButton(int buttonIndex) {
    if (gameMode != LISTENING) return;        // Only examine values when game is in play.
    // Guard against entering LISTENING state between a press and a release.
    heardButtonPress = true;
    // Logic for game 2:  We take user input as next color and fall through to normal case.
    if (playerPosition > sequenceLength) {
        currentSequence[sequenceIndex] = buttonIndex;
        sequenceLength++;
        playerPosition++;        // Point past new end of list and trigger restart of matching.
    }

    if (currentSequence[sequenceIndex] == buttonIndex) {    // showButton only if match.
        showButtonPress(buttonIndex);
    } else {
        doStream(soundIds[LOSE_SOUND]);
        if (theGame == 3) {        // Eliminate color that was pressed in game 3.
            activeColors[buttonIndex] = false;
        }
        gameLose();
    }
}

public void showButtonPress(int index) {
    if (index >= 0 && index < TOTAL_BUTTONS) { //if our index exists
        if (buttonPressMap[index] == false) { //and our index on the map is not pressed
            buttonPressMap[index] = true; //press it

            switch (gameMode) { //then enter the gameMode you are in
                case LOSING: //if we are losing play the lose sound
                    doStream(soundIds[LOSE_SOUND]);
                    return;
                case LISTENING:
                    if (currentSequence[sequenceIndex] == index) //if we match our index with the system
                        doStream(soundIds[index]); //play that corresponding sound
                    else
                doStream(soundIds[LOSE_SOUND]); //otherwise play the lose sound
              break;
            default:
              doStream(soundIds[index]); //we play the sound for the corresponding color
              break;
          }
          for (Listener listener : listeners) {
            listener.buttonStateChanged(index); //invalidate to buffer pixels correctly
          }
        }
      }
    }

public void releaseAllButtons() { //we reset our buttonpressmap boolean and turn all buttons off
  for (int i = 0; i < buttonPressMap.length; ++i) {
    buttonPressMap[i] = false;
  }
  for (Listener listener : listeners) {
    listener.multipleButtonStateChanged(); //invalidate once again to ensure smooth bitmap
  }
}

/* releaseButton calculates if the user is correct and then displays the release animation */

public void releaseButton(int buttonIndex) { //logic for releasing a single button
  if (gameMode != LISTENING) return;
  // Guard against acting on a button press that happened before we were LISTENING.
  if (heardButtonPress == false) return;

  heardButtonPress = false;         // Reset our heardButtonPress state.
  mLastUpdate = System.currentTimeMillis();
  if (sequenceIndex < sequenceLength) {
    if (currentSequence[sequenceIndex] == buttonIndex) { // Matched. Continue.
      showButtonRelease(buttonIndex);         // showButtonRelease only if match.
      sequenceIndex++; //point to next index
      if (sequenceIndex == sequenceLength) {
        //SCORING SYSTEM LOGIC
        if (hScore == 0) {
          hScore = 10;
        }
        else if (hScore < 0) {
          if (hScore == -10) {
            hScore = 0;
          }
          else {
            hScore /= 2;
          }
        }
        else {
                hScore *= 2;
            }
            e.setText("Score: " + hScore);
            if (hScore > ultimateHs) {
                ultimateHs = hScore;
                saveHighScore();
                isAHS = true;
            }
            if (sequenceLength < totalLength) {  // add one more.
                if (theGame == 2) {  // in game 2, user adds next item in sequence
                    if (playerPosition > sequenceLength) {
                        playerPosition = 1;
                        sequenceIndex = 0;        // we added one. Now restart matching sequence.
                    } else {
                        playerPosition++;        // set the stage for adding to sequence on next button press.
                    }
                } else {
                    sequenceLength++;
                    playerPosition = 1;
                    scaleBeepDuration(sequenceLength);
                    currentSequence[sequenceIndex] = getRandomColor(); //get another color
                    gameCycle(); //cycle the game to play next color
                }
            }
        } else {
            playerPosition++;
        }
    } else {
        if (theGame == 3) {        // Eliminate color that was pressed in game 3.
            activeColors[buttonIndex] = false;
        }
        gameLose();
    }
}
}

public void playLast() { //used with our Play Last button to play the current sequence again
    for (int index = 0; index < 4; index++) {
        showButtonRelease(index); //In case user was fast on the draw: Reset all buttons.
    }
    gameMode = PLAYING;
    sequenceIndex = 0;
    update();
}

//Loading the High Score
public int loadHighScore() {
    SharedPreferences sharedPreferences = getSharedPreferences("MyData", Context.MODE_PRIVATE);
    int hs = sharedPreferences.getInt("HighScore", 0);
```

```
        if (hs == 0) {
        }
        else {
            ultimateHs = hs;
        }
        return ultimateHs;
    }

    //Saving the High Score
    public void saveHighScore() {
        if (ultimateHs >= hScore) {
            editor.putInt("HighScore", ultimateHs);
            editor.commit();
        }
    }

    @Override
    protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        model.saveState(outState);
    }

    //default constructor to fix error in android manifest file
    public MemTrain() {
    }

    @Override
    public void onBackPressed() {
        startActivity(mainmenu);
    }
}
```

HighScore.java

```
package daniel.memtrain;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class HighScore extends Activity {
    private int highscore = 0;
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_high_score);

    loadHighScore();

    TextView screenText = (TextView)findViewById(R.id.hsText);
    String fontPath = "font/arcadepix.TTF";
    Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
    screenText.setTypeface(tf);
    screenText.setTextSize(30);
    screenText.setText("High Score: " + highscore);
}

public void loadHighScore() {
    SharedPreferences sharedPreferences = getSharedPreferences("MyData", Context.MODE_PRIVATE);
    int hs = sharedPreferences.getInt("HighScore", -999);

    if (hs == -999) {

    }
    else {
        highscore = hs;
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_high_score, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

```
}
```

Help.java

```
package daniel.memtrain;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.widget.ImageButton;
import android.widget.TextView;


public class Help extends Activity {
    ImageButton credits;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);

        TextView help = (TextView)findViewById(R.id.textView);
        TextView help2 = (TextView)findViewById(R.id.textView2);
        String fontPath = "font/arcadepix.TTF";
        Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
        help.setTypeface(tf);
        help2.setTypeface(tf);

        final Intent creditsPage = new Intent(this,Credits.class);

        credits = (ImageButton) findViewById(R.id.credits);
        credits.setOnTouchListener(new View.OnTouchListener(){
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()){
                    case MotionEvent.ACTION_DOWN:
                        credits.setImageResource(R.drawable.creditshelpbuttontwo);
                        break;
                    case MotionEvent.ACTION_UP:
                        credits.setImageResource(R.drawable.creditshelpbuttonone);
                        startActivity(creditsPage);
                        break;
                }
            return false;
          }
        });

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_help, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

Credits.java

```
package daniel.memtrain;

import android.app.Activity;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;


public class Credits extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_credits);
        TextView credits = (TextView)findViewById(R.id.textView3);
        String fontPath = "font/arcadepix.TTF";
        Typeface tf = Typeface.createFromAsset(getAssets(), fontPath);
        credits.setTypeface(tf);
        credits.setGravity(Gravity.CENTER);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_credits, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/background_hd"
    android:gravity="center"
    android:baselineAligned="false">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView"
        android:src="@drawable/memtrainlogo" />

    <ImageButton
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/startGame"
        android:layout_gravity="center_horizontal"
        android:src="@drawable/sgameone"
        android:background="@android:color/transparent"/>

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/highScore"
        android:src="@drawable/highscoreone"
        android:background="@android:color/transparent" />

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/help"
        android:layout_gravity="center_horizontal"
        android:src="@drawable/helpone"
        android:background="@android:color/transparent"/>

</LinearLayout>
```

game_activity_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@drawable/background2">

<daniel.memtrain.ButtonsMain
    android:id="@+id/button_grid"
    android:layout_alignParentTop="true"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />


<TableLayout android:id="@+id/button_table"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:stretchColumns="1,3"
    android:background="#0052dca8">
    <RelativeLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:gravity="center">

        <ImageButton
            android:id="@+id/start"
            android:src="@drawable/playgamesone"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:background="#0052dca8"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />

        <ImageButton
            android:id="@+id/last_button"
            android:src="@drawable/playgamelone"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:background="#0052dca8"
            android:layout_alignParentTop="true"
            android:layout_toRightOf="@+id/start"
            android:layout_toEndOf="@+id/start" />

    </RelativeLayout>


</TableLayout>

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/eric_hs"
    android:layout_above="@+id/button_table"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="30dp"
    android:textColor="#ffffff"
    android:textSize="25dp" />

</RelativeLayout>
```

activity_highscore.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="daniel.memtrain.HighScore"
    android:background="@drawable/backgroundhs">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/hsText"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:textColor="#ffffff"
        android:textSize="35sp" />
</RelativeLayout>
```

activity_help.xml

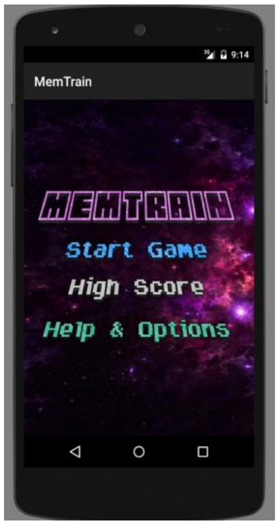
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="daniel.memtrain.HelpActivity"
    android:background="@drawable/background_hd">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:id="@+id/helpText"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:textColor="#ffffff"
        android:textSize="35sp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#fffdffff"
        android:textSize="50dp"
        android:gravity="center"
        android:text="How To:"
        android:id="@+id/textView"
        android:layout_marginTop="21dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#fffff9fc"
        android:textSize="20sp"
        android:gravity="center"
        android:text="After pressing Play Game, you should see a 2 by 2 arrangement of colored buttons. These 4 buttons will
show you a sequence. Pay close attention to this sequence. After seeing the sequence, press the correct buttons to copy the
sequence you saw. Then repeat and see how good your memory is. If you press Play Last, the previous sequence will repeat
but your score will be reduced. Good luck!"
        android:id="@+id/textView2"
        android:layout_centerVertical="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />


    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/credits"
        android:src="@drawable/creditshelpbuttonone"
        android:background="@android:color/transparent"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

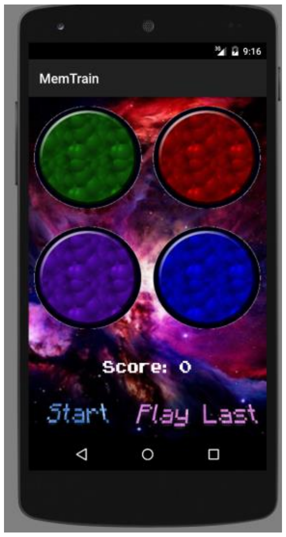
activity_credits.xml

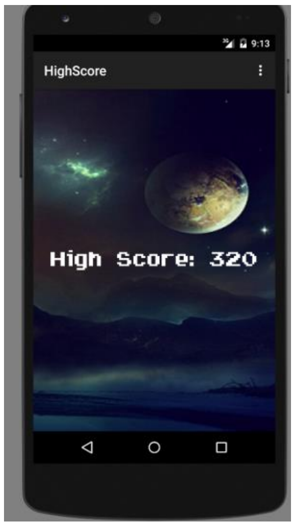
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:background="@drawable/starstwo"
    tools:context="daniel.memtrain.Credits">


    <TextView
```

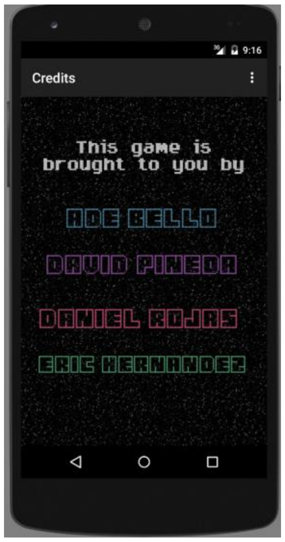
```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This game is brought to you by"
        android:textSize="25sp"
        android:id="@+id/textView3"
        android:layout_above="@+id/imageView4"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="43dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:src="@drawable/danielrojas"
        android:layout_marginBottom="22dp"
        android:layout_above="@+id/imageView3"
        android:layout_alignLeft="@+id/textView3"
        android:layout_alignStart="@+id/textView3" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView3"
        android:src="@drawable/erichernandez"
        android:layout_marginBottom="80dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView4"
        android:src="@drawable/adebello"
        android:layout_above="@+id/imageView5"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="23dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView5"
        android:src="@drawable/davidpineda"
        android:layout_marginBottom="34dp"
        android:layout_above="@+id/imageView2"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Screen Shots








Credits

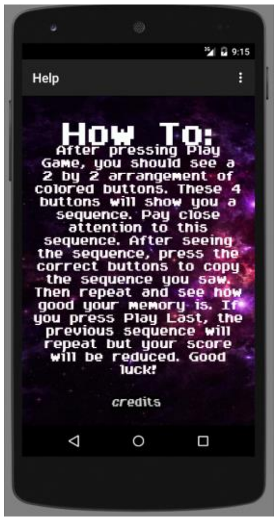
This game is brought to you by

ADE BELLO

DAVID PINEDA

DANIEL ROJAS

ERIC HERNANDEZ


Help

How To:
After pressing Play Game, you should see a 2 by 2 arrangement of colored buttons. These 4 buttons will show you a sequence. Pay close attention to this sequence. After seeing the sequence, press the correct buttons to copy the sequence you saw. Then repeat and see how good your memory is. If you press Play Last, the previous sequence will repeat but your score will be reduced. Good luck!

credits