University of Edinburgh
School of Informatics

The Importance of Being Android

4th Year Project Report
Software Engineering

Daniel Muir

April 3rd, 2013

Abstract: When learning a part in a play, actors need to memorise their lines and know when to deliver them. With this in mind, the aim is to develop an Android application to help actors learn their lines for a play. In addition to simply allowing someone to read the play, the application will act as a digital partner for the user, allowing them to rehearse in various ways.

Acknowledgements

I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this project possible:

My Supervisor, Professor Stephen Gilmore, for his vital encouragement and direction for the project.

A very special thank you to my brother, Andrew Muir, who aided me in finding respondents to the first survey conducted for this project.

All respondents of the survey and evaluative study for their valuable feedback.

All School of Informatics faculty members and staff.

All family and friends

And finally a special thanks to my Mother and Father, for their constant support, encouragement, love and belief in me.

Table of Contents

1. Introduction

1.1 Original description of the Project

When learning a part in a play, actors need to memorise their lines and know when to deliver them. Sometimes the language used in the play is unfamiliar and may necessitate repeated reading and re-reading of the play in order to memorise the lines. It would be useful to have the text of the play available on a phone in order to be able to learn lines whenever is most convenient.

The project was to develop an Android application, primarily to help actors learn lines for a play. For the project, the play; “The Importance of Being Earnest”[1] by Oscar Wilde was loaded into the app and, in addition to simply allowing someone to read the play, users may select a part in the play to learn. In this mode, the text of the play is shown up to their chosen character's next line and the user must click to advance the text. This enforced pause before their line allows the user to try to recall the line, before advancing to see if their recollection was correct.

In addition to making it easy to navigate through the text of the play, the app includes the option of identifying cues in the previous line which prompt the user to help remind them what their next line is. It was required that the user could also add performance notes to the text.

1.2 Reasons for the Project

There were many reasons why developing such a project was something worth doing. First of all, it is easier for the user to learn the lines of their selected play at any time when the script is stored on their device. Although this is already possible with a script on paper, having it stored on a device that is carried around by the user almost permanently, improves the efficiency of learning. It is also made easier for the user to quickly jump to different sections of the play. With the script on paper, the actor must always search through the whole text when looking for their lines. With the app, the user is able to choose which parts of the script they would like to filter.

The most important reason is the idea of creating a “Digital Partner”. Typically an actor will first learn the lines of their play. Once they are confident that they have memorised the play, they will opt to run the lines through with another person so they can act off their performance. Since the app allows the user to select a character in the play, they can re-create this experience without the need of another person. This, therefore greatly enhances the learning experience.

1.3 Finished State of Project

The listed requirements of the project have all been completed successfully. The project has also been extensively tested using the Android testing framework Robotium[2], and has been loaded onto a real device which ensured the application is fit for purpose. Once the state of the application was considered ready for release, an evaluative study was conducted with sample users, who completed a range of tasks and provided feedback on their experience.

1.4 What this report will contain

Section 2 of this report provides the steps taken to prepare for the implementation of the project with the research taken into understanding how an actor rehearses for a role. Research into similar applications is also presented to examine what features were well received and which were poorly implemented, and also to ensure that this project added something new. Finally, initial designs that were constructed to show how the application should look are presented.

Section 3 goes into detail as to how the project was implemented. This section is divided into subsections which detail the work that was done leading up to each project group meeting. Each subsection lists what was achieved and highlights some problems that were encountered as well as how they were solved.

Section 4 explains the steps taken to test the application; presenting both methods using Robotium and a real device. Changes that were made during this part of the development will also be described. Finally, a brief evaluation of the final code metrics will also be described.

Section 5 involves the evaluation process. This section shows how the evaluative study was carried out; detailing both what users would be asked to do and why. Finally the results that were gathered will be presented and analysed.

Finally, Section 6 concludes the report. First some ideas for further work in the future are presented, and then a conclusion of whether the main goals of the project were completed successfully.

2. Research and Design

2.1 Research

Before beginning to think about design and implementation of the project, it was decided that time should be spent on some background reading. This was to understand how actors prepare for a role in a play, and to determine if the original specification of the project should be expanded to include additional features and functionality which had not been proposed initially. Also some time was taken to establish if there were any other applications currently on the market which already helped actors to rehearse. These applications, once identified, were examined to ensure that this project would deliver some unique qualities that were, at present, non-existent in the Android market.

2.1.1 Research into how actors rehearse

Recording was a technique that was described as being an extremely useful tool for learning lines.[3] This was something that would work well with the application, as the devices it would be used on normally have a recording feature built in. An extension of this idea was to allow the user to store the audio of their play onto their device, thus hearing the other characters of the play while rehearsing.

A few other ideas were agreed upon after discussion with the project supervisor. It was decided that the script of the play should be stored in a SQLite Database. The main focus of the app was to allow the user to quickly retrieve different parts of the play. Since databases allow for queries to quickly retrieve the desired data, this seemed like an effective data structure to use.

When an actor is rehearsing for their play and they have difficulty remembering their line, the prompter will reveal some words and the actor can continue without breaking rhythm. This is a method that would also be included, so again, rehearsing with the app would closely mirror rehearsing with others around you.

Finally the user would have the ability to display only their own lines for their selected character for ease of learning, and statistics were to be recorded to provide some feedback to the user e.g. number of views for different parts of the script.

2.1.2 Research into other existing applications

At the time of writing this report, there were a few applications available on the market that provided similar functionality. Probably the most successful app was Rehearsal.[4] Rehearsal allows users to load a PDF copy of the script into the app and the user can highlight and blank out text by drawing on the screen. Recording and adding notes were also available among many other less significant features. The other applications available provided similar functionality, but Rehearsal seemed to be the most popular among users.

Although many of these applications existed, they were exclusively available to Apple devices. When searching the Android Market, only two apps were found that were aimed at helping actors rehearse. Rehearsal Assistant/Voice Recrd[5] only allowed the user to record themselves, but not actually load a script. Line Please[6] seemed to be the only Android application that was similar to this app. In Line Please, the user can read from a script but each line must be manually typed into the app. However, it was not very well received by those who downloaded the application, as it crashed a lot of the time and the user is required to type lines one at time, rather than directly load a script.

Crucially, none of these similar applications shared the feature of hiding lines until the user chooses to progress. That, along with the availability of only a small number of apps on the Android market, would ensure that this project would offer something new to actors wanting an application to rehearse.

2.2 Functional and Non-Functional Requirements

Functional Requirements

- The user should be able to load any script of their choosing into the application.
- The user should be able to select any character from the script to rehearse as and also select a part of the play to begin rehearsing from.
- Lines of the text should be hidden until the user chooses to proceed through the script.
- The user should be able to add notes to any of the lines of the script.
- The user should be able to record themselves rehearsing and also play back audio of other characters in the script.
- The application should have the flexibility of allowing the user to toggle cue words, stage directions or only viewing their character's lines.
- The application should provide feedback to the user in the form of statistics that record their activity.

Non-Functional Requirements

- Loading a script onto a device should be a simple task so that users are willing to re-use the application when preparing for new roles.
- Retrieving lines of the script from the database should be executed quickly so that the user will prefer using this format for rehearsing (rather than using a hardcopy, PDF, etc.)
- The application must be robust in that unexpected actions from inexperienced users can be handled efficiently. Users should be corrected when mistakes are made.
- Overall the response time of performing tasks with the app should be as quick as possible.
- The application should be easy to learn how to use and also importantly easy to keep using. It should be accessible to any demographic.

2.3 Tools required for development

Once the planning was completed and the requirements were identified, a list was made noting which hardware and software tools would be needed to successfully develop the project.

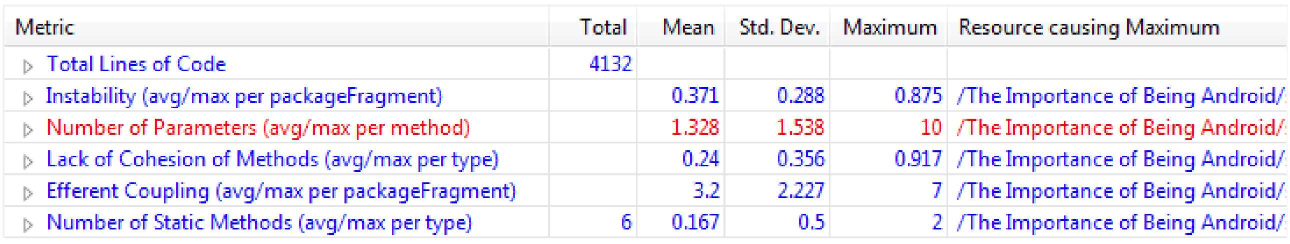
Software Tools:

- Eclipse IDE to develop the software.
- Android development plugin for Eclipse.
- Image editing software for creating screen layouts.
- Word Processing software for writing documents.
- Version Control for backing up software (GitHub).
- Robotium framework for testing.
- Dropbox for backing up files and transferring between computers.
- Google Drive for creating surveys to be distributed and also for creating presentations.

Hardware Tools:

- Computer compatible with all of the Software Tools listed above.
- A real Android device for testing. Multiple preferred.

Metrics plugin for Eclipse was another tool that was used.[7] While not specifically required to complete the project, it was useful in providing statistics such as the average number of lines of code per method. Statistics that were considered good were presented in blue font, while bad coding practice was highlighted in red. Following this Metrics tool, the completed project should follow good programming practice.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum
Total Lines of Code	4132				
Instability (avg/max per packageFragment)		0.371	0.288	0.875	/The Importance of Being Android/
Number of Parameters (avg/max per method)		1.328	1.538	10	/The Importance of Being Android/
Lack of Cohesion of Methods (avg/max per type)		0.24	0.356	0.917	/The Importance of Being Android/
Efferent Coupling (avg/max per packageFragment)		3.2	2.227	7	/The Importance of Being Android/
Number of Static Methods (avg/max per type)	6	0.167	0.5	2	/The Importance of Being Android/

Figure 2.1: Sample of Metrics plugin for Eclipse

For the project build target, API level 10, or more specifically 2.3.3, was chosen. According to data provided by Android Developer website, API level 10 has the biggest share of the user market with 43.9%. A total of 90.2% of the Android market have a minimum of API level 10, and would be able to use the application.[8]

Finally, the app would require permissions to use the microphone and the SD card of a device. The microphone would allow the user to record themselves while the SD card would allow the user to save scripts to their device which could then be loaded by the application, and also store recordings they have created. The app would not require any permissions to connect to the internet.

2.4 Design

Before moving onto implementation, some basic designs were constructed to show how the application should look and perform. The designs referenced below can be found in Figure 2.2.

2.4.1 Home Screen

The Home Screen is the starting point for the user when they load the application. From here they would be able to go to any of the other screens in the app. To do this, the user scrolls through the text on screen which can be done by pressing the up and down arrows.

2.4.2 Options Screen

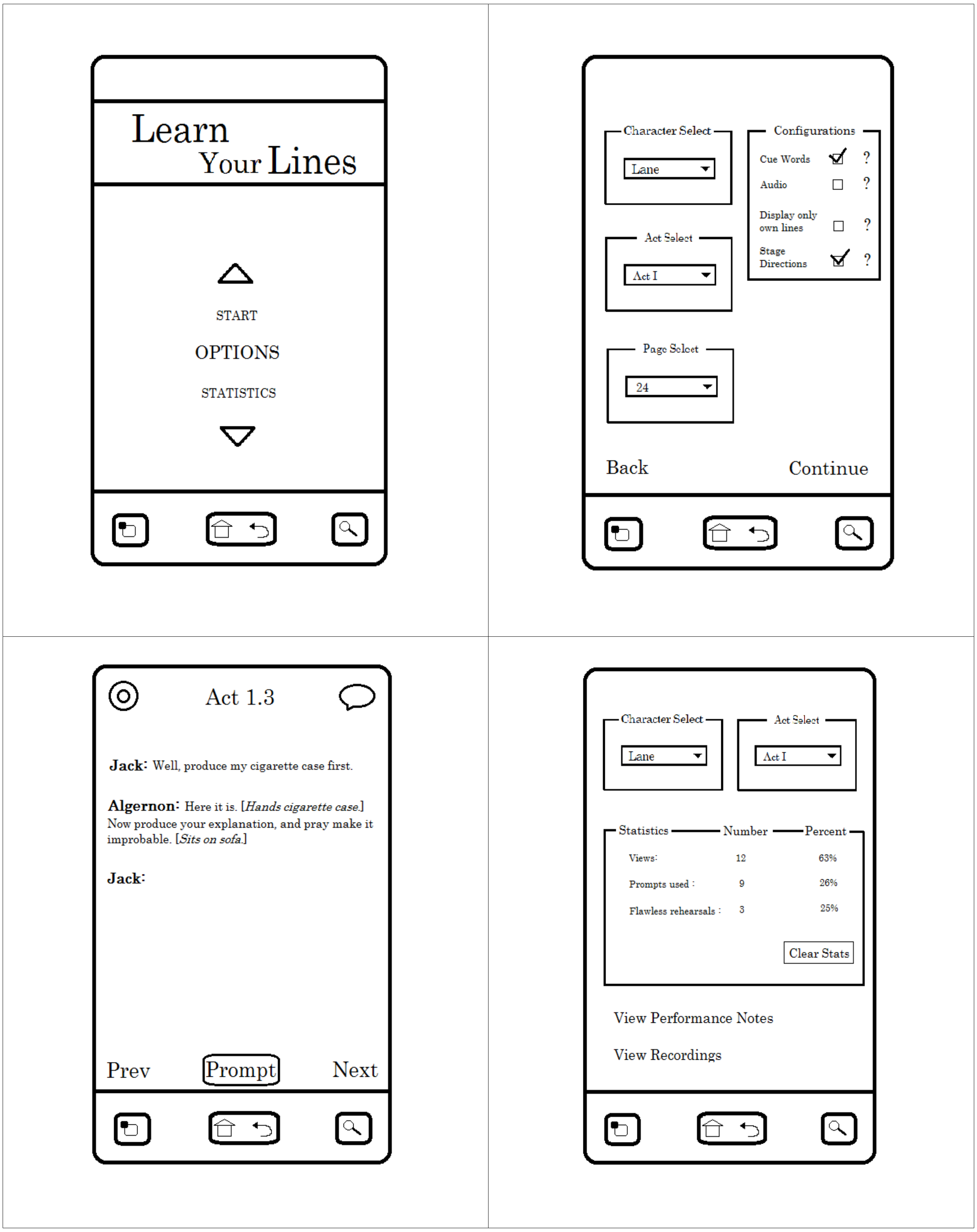
The Options Screen is where the user would be taken after pressing “Start” on the Home Screen. Here the user can select which character they would like to rehearse, the act at which they would like to begin and also the page within that act. One page equates to roughly one A4 size of a sheet of paper. The user could also toggle “Cue Words”, “Audio”, “Own Lines” or “Stage Directions” on or off. When the user presses “Continue”, they would proceed to rehearse their lines, based on their configurations.

2.4.3 Main Screen

The Main Screen is where the user would spend most of their time. To reveal the selected character's current line, the user can choose “Next”, and more lines will be revealed to them until their character's next line. Pressing and holding on “Next” will allow them to proceed to the next page. Similarly, “Prev” will work in the same way, but in the opposite direction. “Prompt” will reveal the next hidden word from the current hidden line. The “Recording” button can be found at the top left of the screen. By pressing this, the device will begin recording the user's dialogue. At the top right is the “Performance Notes” button. The user can create a new note by pressing this button.

2.4.4 Statistics Screen

The user may view their stats at any time on the Stats Screen. Similar to the Options Screen, the user can select a character, an act or a page and view recorded stats based on their selection. The stats show the number of views, the prompts used, and “Flawless Rehearsals”. Flawless Rehearsals are the number of times the user managed to progress through the current selection without requiring a prompt. The percentage stats show how much of the total count the user's current selection represents. For example, Figure 1 has the value 3 and 25% for the Flawless Rehearsal statistic. This means that for the whole app in total, the user has made twelve Flawless Rehearsals. This indicates to the user they are not doing too well when rehearsing this current selection, and should work to improve. The user can also view notes created and recordings made.



Configurations		
Cue Words	✓	?
Audio	☐	?
Display only own lines	☐	?
Stage Directions	✓	?

Jack: Well, produce my cigarette case first.

Algernon: Here it is. [*Hands cigarette case.*] Now produce your explanation, and pray make it improbable. [*Sits on sofa.*]

Jack:

Statistics	Number	Percent
Views:	12	63%
Prompts used :	9	26%
Flawless rehearsals :	3	25%

Figure 2.2: Initial Designs of application

Top: Home Screen, Options Screen
Bottom: Main Screen, Stats Screen

2.5 Project Schedule

The final task before moving on to the implementation was to draw up a project schedule. The schedule (Figure 2.3) lists the tasks that must be completed with their corresponding deadline. The list of tasks conclude with the writing of this report.

Task	Description	Deadline
1	**Finish Implementation.**	**End of Semester 1**
1.1	Design and implement Database.	November
1.1.1	Decide on what fields should be stored into database.	October Week 3
1.1.2	Add filter feature to match user's character and page choice	November
1.2	Add Performance Notes functionality	November Week 2
1.3	Add Cue Words functionality	December
1.4	Add Prompt functionality	December
1.5	Add additional features	End of Semester 2
1.5.1	Record and display stats for the user	December Week 1
1.5.2	Add Recording option	December Week 2
1.5.3	Add Audio option	December Week 3
2	**Test Project**	**February**
2.1	Use Android Testing Framework i.e. Robotium, to test functionality	January Week 3
2.2	Evaluate project on different Android phones to test design	February
3	**Evaluate Project**	**February Week 3**
3.1	Get a range of users to use App	February Week 3
3.1.1	Provide users with tasks to accomplish	February Week 3
3.1.2	Provide users a Questionnaire to note their experience	February Week 3
4	**Write Dissertation**	**April**

Figure 2.3: Project Schedule

3 Implementation

3.1 First Project Group Meeting

In preparation for the first project meeting, the tasks set were to create a basic implementation of the four main screens that had been designed. First was the Home Screen. The only functionality that this screen would have at this point was to allow the user to navigate to the other screens in the app.

Most of the work involved developing the Options screen. As illustrated in the original design, three Spinners were used to filter different parts of the script, while four checkboxes could toggle the different configurations. Since at this point the script of The Importance of Being Earnest had not been loaded into the app, the contents of each Spinner was just hardcoded. Also an “onItemSelectedListener” was added for the act Spinner so the relevant pages would be shown. Finally, a brief help guide was added for each of the checkboxes, explaining the changes if they were enabled.

The Stats screen worked much the same way as the Options screen, with the three Spinners for filtering parts of the script. Some stats were hardcoded to help illustrate the look of the app for the first project group meeting. Finally a “clear” button was added which would reset the stats.

The final screen to be implemented was the Main screen. Again, since at this stage the script was not being loaded, some hardcoded text was inserted to show how revealing a character's next line would function.

Feedback from the first meeting was by and large positive. However, some were concerned that some features were being implemented without any evidence that they would be useful to those using the application.

3.2 Second Project Group Meeting

3.2.1 Crowdsourcing

Before continuing with any more implementation, the next task was to decide how to show that the features being implemented were going to be useful. To do this, it was required to obtain input from those involved in acting.

To gather the potential user's opinions, a survey was created using Google Docs. The user was asked to rate each of the following features:

- Filtering
- Prompt (reveal a word from hidden line)
- Cue Words (highlight words to hint to the user their hidden line)
- Recording
- Audio
- Performance Notes
- Statistics

Each of the above features could be rated as; ***very helpful***, ***helpful***, ***don't know***, ***unhelpful*** or ***very unhelpful***. Finally respondents were asked if they use any techniques that were not covered by the features they were asked to rate.

3.2.2 Crowdsourcing Results

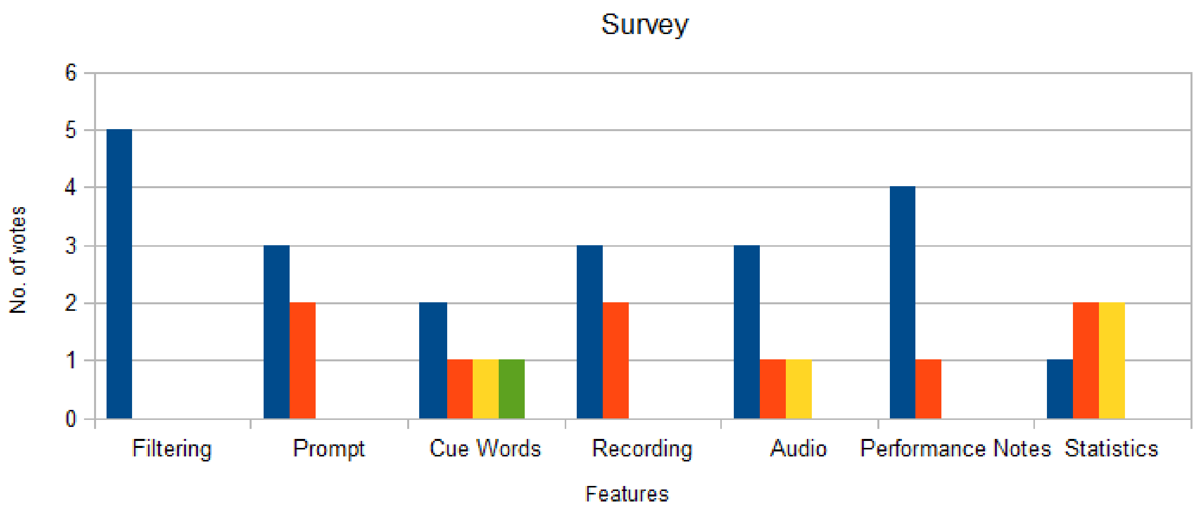
Five responses were made to the survey and the results can be seen in the graph on the next page (Figure 3.1). Overall the response was good to all of the features that were rated. Some of the features were a little debated, like the Statistics feature for example. However, all of the features received an average rating of at least **help*ful***. This justified that all the listed features should be developed.

When respondents were asked to provide their own techniques for rehearsing, three replied with the following interesting comments:

- "I think if the app could link to an osx native app that allows you to print out your scripts with the notes and additions made in the app that would be useful as well. My main problem is having multiple scripts with multiple versions of notes. It would be great to be able to print out a fresh one as rehearsals progress."

- "random lines- the app gives random lines / the final words of lines of the other characters, not in order that they come in the script. This avoids the actor to learn 'chunks' of the script, and provides a more in depth knowledge of their cue lines."

- "I personally find it easier to learn lines to songs, or tunes. So maybe have some tunes or songs that you can choose from. Just an idea. The rest of it sounds good though."



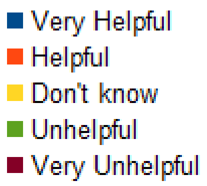


Figure 3.1: Graph of results obtained from crowdsourcing survey

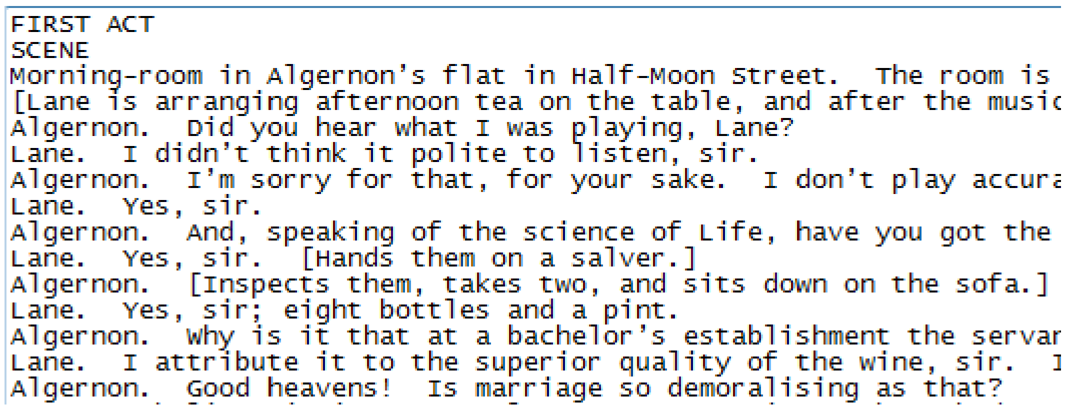
3.2.3 Implementation for Second Meeting

3.2.3.1 Database creation

In preparation for the second project meeting, much more development was completed. The first thing created was the database which stored the script. The play was loaded into a standard text file (Figure 3.2), which was then placed in the Assets folder in the Project Directory. The entirety of the text file was then read by the app and the data was stored into the relevant parts of the database.

In order to make sure the database storing the script is populated correctly, the following strict rules must be adhered to when creating the text file:

- One line in the text file represents one line in the script.
- All stage directions are within square brackets.
- The name of the character must be immediately followed by a period. This is so the character speaking the line and the line itself can be differentiated.
- When there is a new act in the play, there must be a line on its own which has “ACT” as its last word.



```
FIRST ACT
SCENE
Morning-room in Algernon's flat in Half-Moon Street.  The room is
[Lane is arranging afternoon tea on the table, and after the music
Algernon.  Did you hear what I was playing, Lane?
Lane.  I didn't think it polite to listen, sir.
Algernon.  I'm sorry for that, for your sake.  I don't play accura
Lane.  Yes, sir.
Algernon.  And, speaking of the science of Life, have you got the
Lane.  Yes, sir.  [Hands them on a salver.]
Algernon.  [Inspects them, takes two, and sits down on the sofa.]
Lane.  Yes, sir; eight bottles and a pint.
Algernon.  Why is it that at a bachelor's establishment the servar
Lane.  I attribute it to the superior quality of the wine, sir.  I
Algernon.  Good heavens!  Is marriage so demoralising as that?
```

Figure 3.2: Section of the script stored
in the text file

The file is read line-by-line and and we keep a record of the line count, act count and page count. The page count is incremented every 23 lines. This represents roughly one A4 page. We then split the current line into an array of Strings, separated by spaces. The first String in the array is examined to determine if it is a character in the script. A String is a character if it does not contain an opening square bracket or is not entirely uppercase. Otherwise it is a stage direction. We then perform checks for special cases i.e. the character name is more than one String or more than one character is speaking the same line. By using the identifier (the period) we should successfully extract the character speaking the line and the line itself by the time we are ready to enter the data into the database. The code for parsing the file can be found at Appendix A.

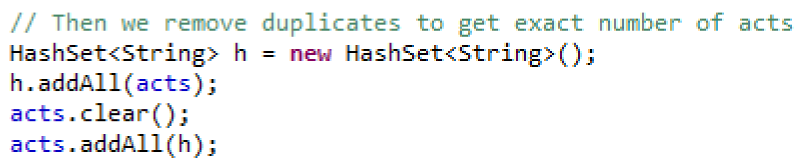
As each line of the text file was read, a new row was created and entered into the database. The following is a list of the data that was contained in each row:

- A count of the line number
- Character speaking the line
- The line itself
- The act the current line was in
- The page the current line was in.
- Whether a note had been created for this line (Initialised to “N”)
- Whether an audio file was associated for this line (Initialised to “N”)
- The number of views for the current line (Initialised to 0)
- The number of prompts required (Initialised to 0)
- The number of “Flawless Rehearsals” completed (Initialised to 0)

Now that the database was fully populated, the script could be filtered based on the user's configurations.

3.2.3.2 Filtering Script on screen

Now that the database was populated, work could begin on displaying the script to the user. The first task was to populate the Spinners on the Options screen dynamically. For each Spinner, the database was read to obtain all the different characters, acts and pages available. Duplicates were then removed and the remaining items were sorted. To remove duplicates, a HashSet was used (Figure 3.3). The list of all the items added to the HashSet were automatically removed, and the result was returned into the list.



```
// Then we remove duplicates to get exact number of acts
HashSet<String> h = new HashSet<String>();
h.addAll(acts);
acts.clear();
acts.addAll(h);
```

Figure 3.3: Example of the HashSet being used to remove duplicates

The character Spinner was sorted based on the number of lines each character had. A HashMap was set up which stored the character as the key. The count of how many times that character appeared in the original list of duplicates was stored as the value. The characters were then sorted by repeatedly removing the <key, value> pair with the maximum value and storing the key (the character) into another list, until the HashMap was empty. This meant that the Spinner was ordered by how popular the character was in the script. The code for dynamically populating the character Spinner can be found at Appendix B.

The next step was to display the script to the user based on their selections made in the Options screen. When the user chooses to view the script, a query is made on the database to return all the lines for the page the user selected. A "SimpleCursorAdapter" then displays the character delivering the line and the line itself in a "ListView".

It was at this point a new Spinner was added on the Options screen. This was to allow the user to choose between "NORMAL" and "REHEARSAL" mode:

- NORMAL mode shows the script without any hidden text and statistics are not recorded.
- REHEARSAL mode again shows the script but with the selected character's lines hidden until the user proceeds. When in this mode, statistics would be recorded.

This Spinner was added so the user could first choose to learn the play at their own pace, before selecting a character and testing their knowledge of the script.

3.2.3.3 Problems with REHEARSAL mode

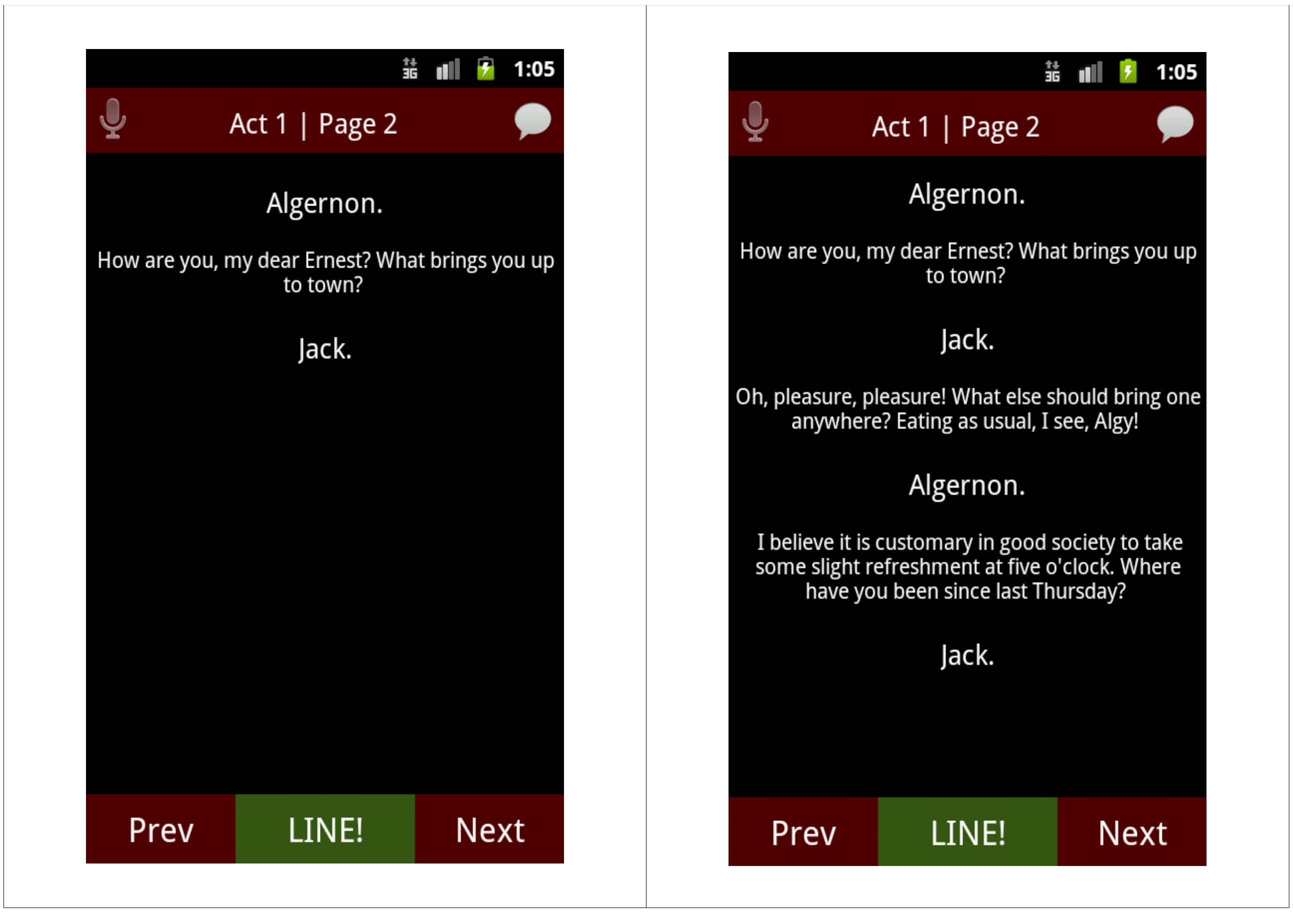
Initially there were some difficulties with implementing the “REHEARSAL” mode that required some changes to be made to the code. The problem was that there was no way to hide the lines from the user. Using the “SimpleCursorAdapter”, we could not limit which lines that were returned from the query could be displayed. To get by this, a custom array adapter was created.

First a “Line” object was implemented. This object was made up of two Strings; the character delivering the line, and the line itself. The custom array adapter (LineAdapter) then populated the ListView with the list of “Line” objects. The custom array adapter had to be created so that the character and line could be extracted from the “Line” object and displayed correctly. A typical array adapter would not be able to do this. The next problem was to work out how to effectively choose how many lines would be displayed to the user.

The first thing that is done when displaying the lines is to obtain a count of how many times the user's selected character appeared before the user chooses to reveal the next line. This is stored in the integer variable “visibleLines”. In other words, the “visibleLines” count increments when the next line is revealed for the user's selected character. We then loop through all the lines that were returned from the query. If the current line that is being read is delivered by the user's character, and the value of “visibleLines” is **more than or equal to one**, then we add the line to the list. The value of “visibleLines” is then reduced by one.

To clarify how this works, let us take an example where currently five lines delivered by the user's character are visible. Here the value of “visibleLines” is six. The value is six because the hidden line is treated as visible. Now as we add the selected character's lines to the list, the “visibleLines” value is decremented and when we eventually break out the loop, there will be six of the selected character's lines in the list. If the user chooses to reveal another line, the value of “visibleLines” is seven and there will six of the selected character's lines in the list, and so on.

Finally, a String “command” is passed as a parameter in this method. This parameter tells the method if we are advancing through the page revealing lines, or moving backwards and hiding lines. The value of this parameter is either “forward” or “back”. The algorithm remains largely the same if “back” is the value passed, except we add lines to the list while “visibleLines” is **more than or equal to three**. Taking the example of when “visibleLines” is equal to six again; when we break out of the loop of adding lines, there will be four of the selected character's lines in the list. The code for this can be found at Appendix C.



Algernon.

How are you, my dear Ernest? What brings you up to town?

Jack.

Algernon.

How are you, my dear Ernest? What brings you up to town?

Jack.

Oh, pleasure, pleasure! What else should bring one anywhere? Eating as usual, I see, Algy!

Algernon.

I believe it is customary in good society to take some slight refreshment at five o'clock. Where have you been since last Thursday?

Jack.

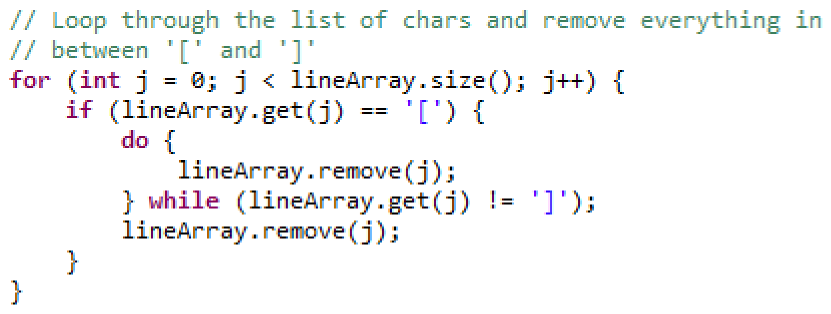
Figure 3.4: Illustrating the effect of the user revealing a hidden line

3.2.3.4 Filtering out Stage Directions

Once all the lines have been added to the list, if the user has decided to remove stage directions then we need to filter them out before displaying the script. We loop through all the lines in the list and check for the following:

- The value of the character is equal to “STAGE”.
- Otherwise check the line for an opening square bracket.

When the character value is “STAGE”, this is an indication that the entire line is a stage direction. Thus we can remove the entire item from the list. Otherwise we split the line into an array of chars. We then loop through each char until we find an opening square bracket. If we find one, then we continuously remove the chars from the array until we encounter a closing square bracket. Finally, the array of chars is built back together to make a String, and the line is updated.



```java
// Loop through the list of chars and remove everything in
// between '[' and ']'
for (int j = 0; j < lineArray.size(); j++) {
    if (lineArray.get(j) == '[') {
        do {
            lineArray.remove(j);
        } while (lineArray.get(j) != ']');
        lineArray.remove(j);
    }
}
```

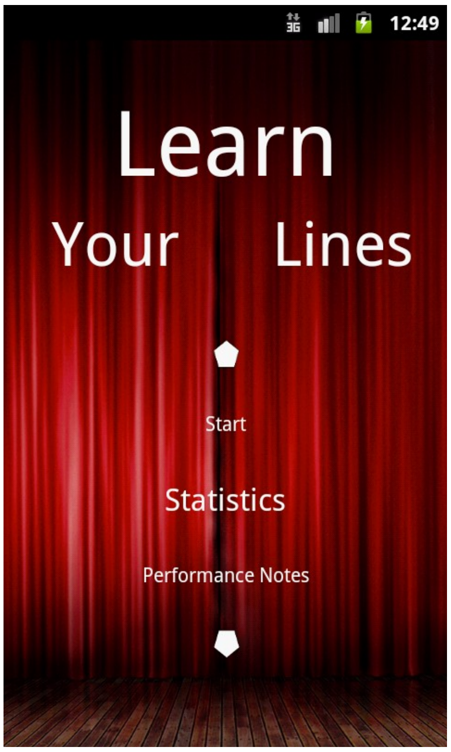
Figure 3.5: Stage directions being removed from a line

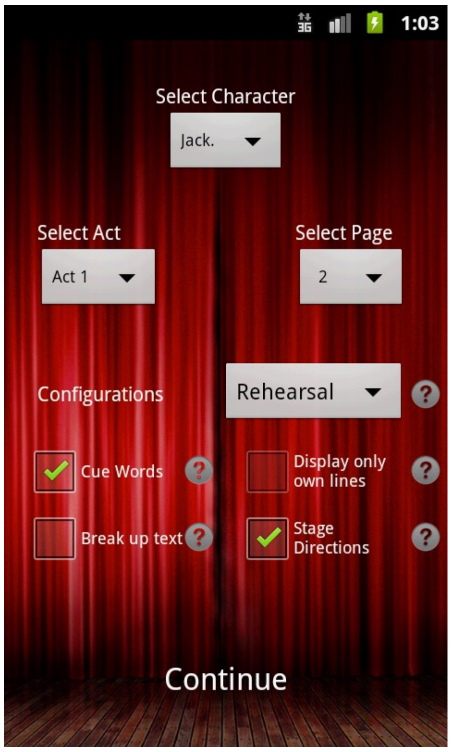
3.2.3.5 Own Lines Feature

The user could also now toggle whether they wanted to display their selected character's own lines only. If this was toggled, the database would be queried to return the selected page and only the selected character's lines on that page. It should be noted that the character selection Spinner on the Options Screen is always disabled unless “REHEARSAL” mode was selected or “Display own lines only” was toggled on. Own lines could only be toggled if “NORMAL” mode was selected.

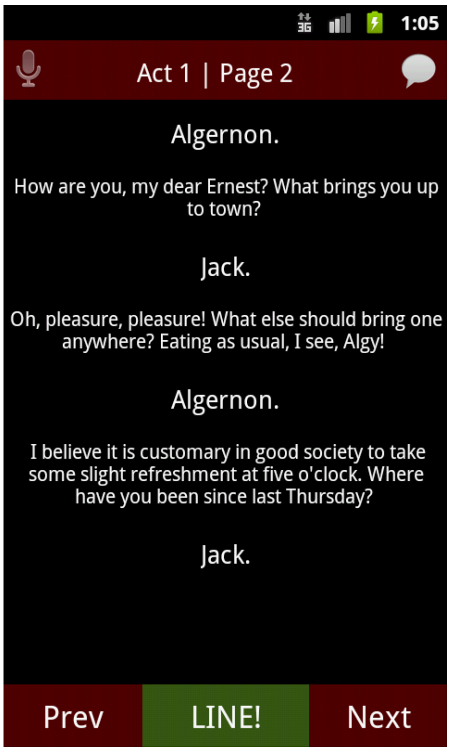
3.2.3.6 Adding the Prompt Feature

The final feature to be added before the second project group meeting was the prompt feature. The prompt feature would reveal a hidden word from the current hidden line to help the user remember that line, rather than instantly revealing it in full. An integer variable “visibleWords” was set up to keep count of how many words were currently visible. This variable was initialised to one, and was reset to this whenever we reveal or hide a line, or when we move to a new page. If the user decides to reveal a new word, we repeatedly add words from the beginning of the hidden line until we reach the limit given by the “visibleWords” value. We then update our list of lines with our “new” line and redisplay the script. Finally, “visibleWords” is incremented. The implementation of this can be found at Appendix D.





Select Character: Jack.
Select Act: Act 1
Select Page: 2
Configurations: Rehearsal
✓ Cue Words
☐ Display only own lines
☐ Break up text
✓ Stage Directions



Algernon.

How are you, my dear Ernest? What brings you up to town?

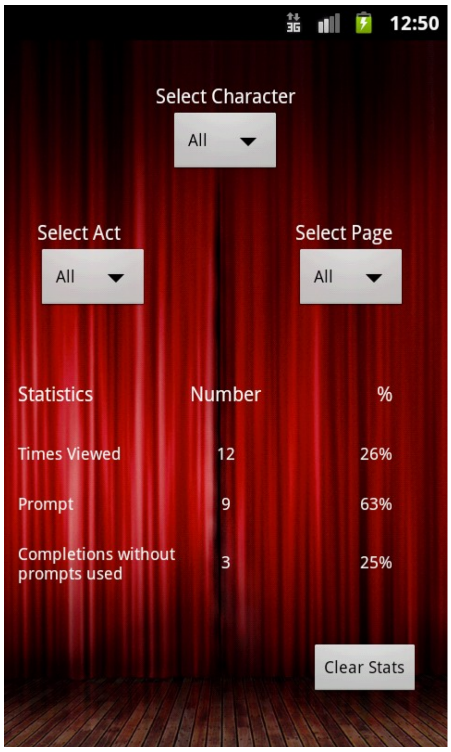
Jack.

Oh, pleasure, pleasure! What else should bring one anywhere? Eating as usual, I see, Algy!

Algernon.

I believe it is customary in good society to take some slight refreshment at five o'clock. Where have you been since last Thursday?

Jack.



Select Character: All
Select Act: All
Select Page: All

Statistics	Number	%
Times Viewed	12	26%
Prompt	9	63%
Completions without prompts used	3	25%

Figure 3.6: State of app before second project group meeting

Top: Home Screen, Options Screen
Bottom: Main Screen, Stats Screen
(Note: Stats are hardcoded)

3.3 Finishing Implementation

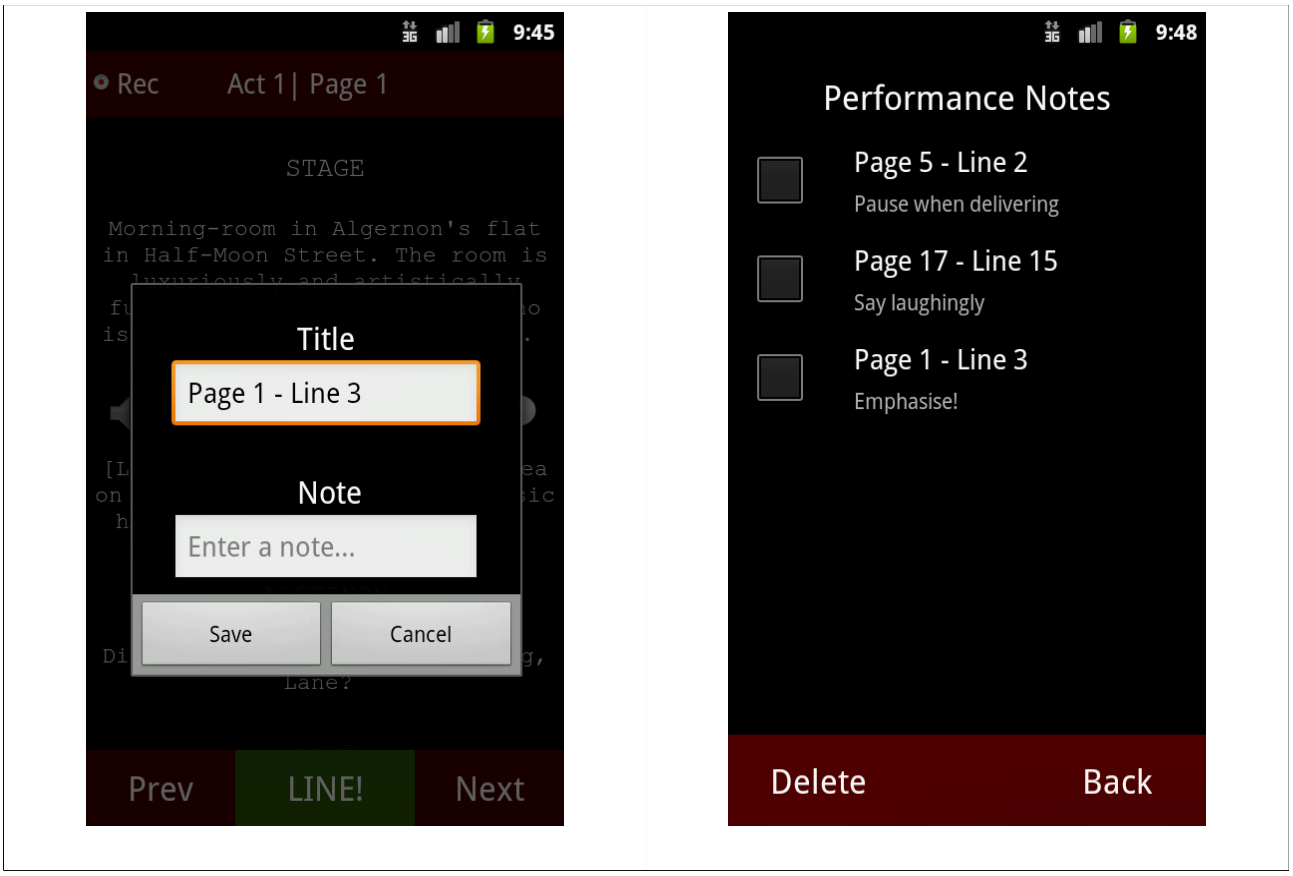
3.3.1 Performance Notes

Now that the application could function on a basic level, the rest of the semester was used to implement the remaining features. The next function that was created was the Performance Notes feature. The original idea was to add the note as a column in the database table for the script. However it would not be possible, or at least be very difficult, to add more than one note to one line in the script. Thus it was decided that a new table would be created that would handle the notes created by the user. The data that was entered into this table was:

- Line Number (so we know where the note was created)
- Title (Page and line number specifically for that page as default. Could be changed if the user wished to do so).
- Note

Also in the original design, the button to create a new note was at the top right of the screen. This was changed so the user could press and hold on their chosen line to bring up a context menu, from where the note could be created. This made it much easier to add the note to the relevant line as the position of the user's selection in the ListView could be obtained. A dialog then appears and the user can enter the text for their note. The only requirement for creating a note was that both the title and note fields must contain at least one character. When the user saved their note, an icon was then revealed next to the relevant line. Pressing on this icon would display the list of notes associated with the selected line.

Finally a new screen was added that would display all the notes the user had created. The notes were loaded in a standard ListView using a “SimpleCursorAdapter”. From here the user can also edit and delete their notes. Each row in the list contained the title and text of each note as well as a checkbox. The checkbox could be used to select one or more items and delete them.



Morning-room in Algernon's flat in Half-Moon Street. The room is

Title: Page 1 - Line 3

Note: Enter a note...

Page 5 - Line 2
Pause when delivering

Page 17 - Line 15
Say laughingly

Page 1 - Line 3
Emphasise!

Figure 3.7: User creating a new note
and Performance Notes screen

3.3.2 Refactoring the Database

After creating the performance notes feature, a lot of refactoring for the database needed to be done. This was because of the way the database adapter was set up. New adapters were being created in different places and the app repeatedly crashed due to conflicts with opening and closing the adapter, and not destroying them properly. A solution was found after some time was spent learning how to correctly use the adapters.[9] A new class was created (“LinesApp”) which extended “Application”. This class would be initialised when the application started, and would close when the application ended. In this class a single adapter was created and opened for both the script and the notes tables. Now any subsequent classes that required the use of any of the adapters could retrieve them from the main class. This meant that there was no need to worry about opening and closing adapters as the single adapter would be closed when the application terminated.

3.3.3 Recording

The next major feature that was developed was the recording feature. Some time was spent on learning how to setup recordings and play them back. A tutorial on YouTube provided instructions on how to do this.[10] First a “MediaRecorder” was initialised and setup to output the data retrieved from a recording into a temporary file. Similar to how the performance notes feature was implemented, the user presses a button which reveals a dialog from where they can begin recording, while a “Chronometer” displays the duration. Finally, once finished, a new dialog is revealed, which allows the user to preview and save their recording. A “MediaPlayer”, which allows playback of the recording, is initialised in the same way as the “MediaRecorder” which sets the data source as the newly created temporary file. Only alphanumeric text is allowed when entering a name for the recording to ensure the file is saved correctly. When the audio file is saved, the data from the temporary file is copied to a new file with the user's chosen filename and the temporary file is deleted. Finally a progress bar was displayed which increments its position as the audio is played back.

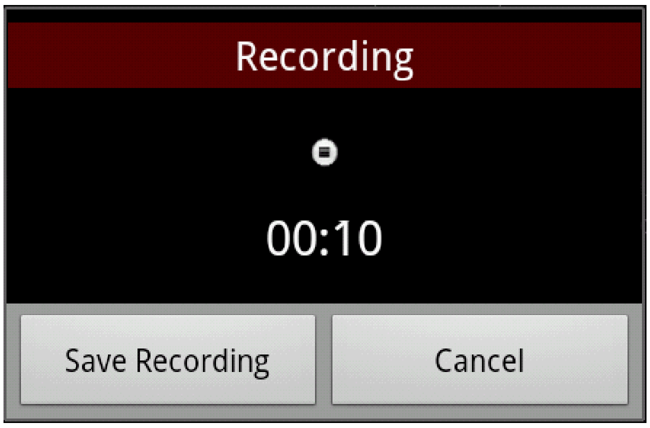


Figure 3.8: The first dialog that appears to record the user

Again like the performance notes, all recordings can be viewed on a new screen in a ListView. A new class was created (“Recording”) which stored the name and duration of an audio file. A list of Recordings was setup and displayed employing a “RecordingAdapter”. This adapter was used in the same way as the one used for displaying the lines of the script (“LineAdapter”). Here the recordings can be renamed, deleted and importantly, listened to.

3.3.4 Audio

The next task was to develop the audio functionality. Originally the plan was to allow the user to download an audio version of their script and listen to that. However this would not be possible, as audio may not exist for some scripts. It could also lead to legal problems in obtaining the audio. To get past this, it was decided that it was up to the user to make the recordings themselves. They could then apply the audio files to lines in the script.

To apply a recording, the user selects a line to bring up a context menu. From here, the user chooses “Apply Recording” and is taken to the Recordings screen. Once the user picks out their recording, the selected line in the database is updated. The audio column in the database is changed from the initial value of “N” to the filename of the selected recording. If the user wants to play back the recording which has been applied to a line, the database retrieves the filename and is set to the “MediaPlayer”.

3.3.5 Statistics

The next task was to record statistics as the user navigated through the script while in REHEARSAL mode, then display them correctly to the user when they chose to view them. The first decision to be made was when to increment the counters for each of the statistics' categories. The total view count was incremented whenever the page was switched and when the screen that displays the script is first initialised. The prompt count was incremented whenever the user pressed the prompt button. Finally the counter that checked for a complete rehearsal of a page was incremented whenever the page was switched. For this value to increase, a boolean “promptsUsed” had to be false (set to true whenever a prompt is used) and the user also had to have the last line on the page visible.

When displaying the statistics to the user, firstly the total of each is counted up by going through each line in the script and incrementing the value. The database is then queried, again depending on the user's selection, and the total is counted up again in the same way. Finally the percentage is calculated by dividing the count of the user's selection by the total. If the total of a statistic is equal to zero then the value is incremented. This is so we do not divide by zero. The code for this can be found at Appendix E.

3.3.6 Cue Words

The Cue Words feature was also developed. This compared two lines that followed one another, checked for duplicate words, and coloured, in red, the duplicates that appeared in the first line. At first the idea was to split each line into array of Strings and then compare each String of each array in a two-dimensional loop. However this had a computational cost of;

$$\boldsymbol{\theta(k(nm))},$$

where n and m is equal to the number of words in the first and second line respectively and k is equal to the total number of lines of the current page minus one.
As a result this algorithm was very costly and slow to execute. After conducting some research, a better solution was found.[11] By splitting the first line and entering it into a Hashmap, we could lookup a value in the Hashmap as we iterated through each String of the second line. This reduced the computational cost to;

$$\theta(k(n+m)),$$

which was an improvement to the execution time.

Once two duplicate words were found, they were checked to make sure they were at least five characters long. This was so words of no value like “and” and “the” could be filtered out. Originally the idea was to find some package that could identify Strings as a noun, and only allow them to be classed as a cue word. However it was difficult to think how this would work with words that did not exist in the dictionary. For example, if the role the user was rehearsing was for some fantasy genre, many of the words in the script would not be recognised. Therefore the current solution, while not perfect, seemed like the better option. Finally, before a line was split into an array of words, all punctuation was removed. This was so there were no problems with comparing Strings i.e. “Hello” should equal “Hello!”, regardless of the exclamation mark.

3.3.7 Randomise

The randomise feature was introduced as a result of the feedback obtained from the survey in section 3.2. If the user decides to randomise, then when they choose to move to either the next or previous page, the page that appears is random. This feature helps to address the problem of “losing your place” on stage where an actor suddenly realises that they have drifted off and do not know quite what is happening in this scene. In this setting, they must be able to recall their line of dialogue from the next line which they hear, even though they have forgotten the surrounding context. The randomise feature helps to recreate this experience.

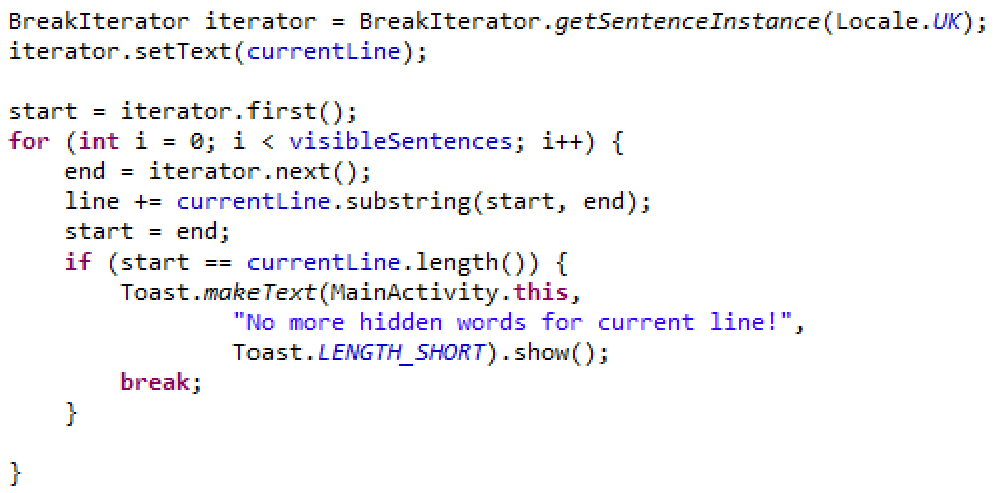
3.3.8 Settings

Finally, development for the first semester was complete with the creation of a Settings screen. On this screen, the user could select which script they would like to view, the number of words to reveal when they press the prompt button and also if they would like to enable auto-playback. The user configurations would be saved to a hidden settings file that would be stored on the phone's SD card.

The list of scripts is displayed in a Spinner that is populated by the script files found on the phone's SD card. If the user decides to load a new script, the existing database would be deleted and a new one would be populated. Since populating a new database took sometime, a new Thread was launched which would display a Progress Dialog, instructing the user to wait.

The user could select to reveal either one, three or five words, or a whole sentence when pressing the prompt button. This was introduced as a result of a problem when allowing the user to only reveal one word at a time. Take the example of when the user is rehearsing a large monologue that is several sentences long. If they were to get almost everything right, but then forget the last sentence, they would then have to press the prompt button many times to eventually reveal the sentence they were having difficulty with. Thus by increasing the number of words revealed with every button press, the prompt feature is therefore better suited for longer lines.

There was a problem when the user chose to reveal a whole sentence as it was difficult to identify what exactly a sentence was. Obviously a period indicates the end of the sentence but there would be some confusion in other instances such as when an ellipses were used. To get around this, a “BreakIterator” was used which successfully splits up a string into sentences.



```java
BreakIterator iterator = BreakIterator.getSentenceInstance(Locale.UK);
iterator.setText(currentLine);

start = iterator.first();
for (int i = 0; i < visibleSentences; i++) {
    end = iterator.next();
    line += currentLine.substring(start, end);
    start = end;
    if (start == currentLine.length()) {
        Toast.makeText(MainActivity.this,
                "No more hidden words for current line!",
                Toast.LENGTH_SHORT).show();
        break;
    }

}
```

Figure 3.9: Sentences revealed from a hidden line

Last was the auto-playback configuration. If this was toggled on, all visible lines on a given page with audio files associated with it would automatically play in sequence. As new lines were revealed, their audio file would also be played if they had one. This was introduced so the user would not have to break their stride to manually play back the audio of another character. Now the user could interact with the other characters as if they were there.

To play back audio files automatically, a recursive algorithm was implemented. First the initial line with an audio file was found by looping through all visible lines in a page and retrieving the line that did not have “N” in its audio column. An integer variable “playbackPosition” was then set to the line number of the page. This was so that when we recalled this recursive algorithm, audio files that had already played back would not play again. This variable was then reset when the user switches page. The audio file that is found is then played back to the user. An “onCompletionListener” was added to ensure that the current audio file was finished, before playing back the next one.

However, another problem occurred here. If there were more than one audio file in a given page, then all the recordings would play back instantly, and not wait until the previous one had finished. To fix this, a listener was added to ensure that the current audio file was finished, before playing back the next one. To play back the next recording, the algorithm would call itself with the updated “playbackPosition” variable. The code can be found at Appendix F.

4 Testing

Now that the features of the application had been developed, some time was devoted to perform some formal testing. Two types of testing were carried out:

- Automated testing using Robotium.
- Manual testing on a real device.

4.1 Robotium Testing

Robotium is a test framework which allows test case developers to write powerful and robust automatic black-box test cases for Android applications. To automate the tests, an object is initialised ("Solo") which performs actions such as clicking on buttons and entering text into EditTexts. Although much testing was carried out during implementation, it was important to include Robotium testing to uncover as many remaining bugs as possible.

```
solo.clickOnButton("Save");
```

Figure 4.1: Example of Solo clicking on a Button

4.1.1 Home Activity

The Home Activity only contained one test. This test ensured that whatever screen the user chose to go to would be the correct one, based on which text they selected. To do this, a random number is generated between zero and five. This number is used to tell the test how many times to click on the arrow to scroll down. The test then selects the chosen text to move to the new screen and it is asserted that we arrive at the expected activity. Using a random number to determine which screen the test navigates to ensures the test will work for any case.

4.1.2 Options Activity

The purpose of the tests here is to check that the different configurations that can be toggled are enabled or disabled correctly, based on the selections the user has made. The tests examine the following:

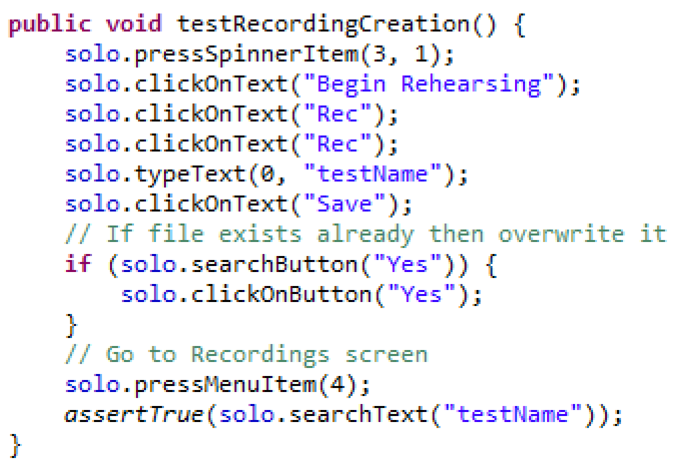
- Character Spinner is disabled when NORMAL mode is selected
- Character Spinner is enabled when REHEARSAL mode is selected
- When cue words are toggled on, own lines is disabled and unchecked
- When own lines is disabled, cue words is disabled and unchecked

Simple Solo commands and assertions were sufficient here to ensure the above statements were true.

4.1.3 Main Activity

The first test written checks for whatever configurations are selected on the Options Screen, are correctly carried over to the next screen where the script is shown. A random number was, again, generated to make the selections. To determine that the correct character was passed through to the Main screen, we scroll to the bottom of the ListView and obtain the last character name shown. It is always the case that the last line (non-visible one) of the first page loaded during REHEARSAL mode is delivered by the chosen character, as the user can never begin from a page where their character does not appear. We also check that this last line is hidden i.e. the String is equal to “”.

Other tests include creating a recording and checking for its existence, and generating some statistics. For the statistics test, controlled actions are performed and it is asserted that the correct values for each statistic is correct.



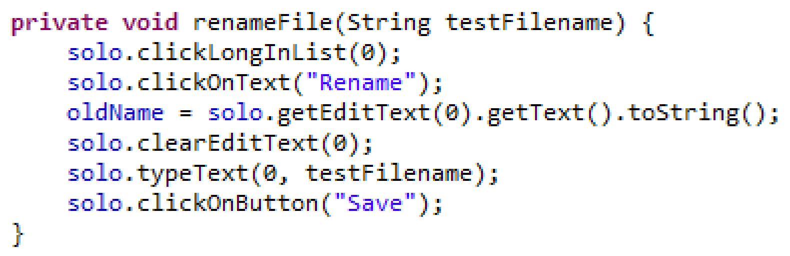
```
public void testRecordingCreation() {
    solo.pressSpinnerItem(3, 1);
    solo.clickOnText("Begin Rehearsing");
    solo.clickOnText("Rec");
    solo.clickOnText("Rec");
    solo.typeText(0, "testName");
    solo.clickOnText("Save");
    // If file exists already then overwrite it
    if (solo.searchButton("Yes")) {
        solo.clickOnButton("Yes");
    }
    // Go to Recordings screen
    solo.pressMenuItem(4);
    assertTrue(solo.searchText("testName"));
}
```

Figure 4.2: Test which creates a recording and checks it exists with the correct filename

4.1.4 Recordings Activity

Since filenames for recordings would only accept alphanumeric characters, the main purpose of the tests here were to rename a recording with different samples and check if the samples were accepted. The following samples were used:

- testName → TRUE
- testName2 → TRUE
- testName! → FALSE



```java
private void renameFile(String testFilename) {
    solo.clickLongInList(0);
    solo.clickOnText("Rename");
    oldName = solo.getEditText(0).getText().toString();
    solo.clearEditText(0);
    solo.typeText(0, testFilename);
    solo.clickOnButton("Save");
}
```

Figure 4.3: Rename a recording with the text given from “testFilename”

4.1.5 Notes Activity

The tests for the Notes Activity were very similar to the Recordings Activity as they both had very similar functionality. Performance Notes however, did not have any restrictions with what text they could be saved as. As a result, the tests only checked that notes were updated correctly with whatever text the user chooses. Another test ensured that should the user delete a note, it no longer existed in the Database.

4.1.6 Statistics Activity

The tests here were aimed at making sure stats would update based on the user's selections. Since the tests for the Main Activity (4.1.3) ensured the correctness of the statistics, the tests here examined the effects of changing the values in the Spinners. A final test ensured that statistics for a selection were correctly reset to zero when a user decides to clear their stats.

4.1.7 Settings Activity

The final set of tests were written for the Settings Activity. Three tests were written that checked:

- The settings file was loaded correctly and the correct items in the Spinners were selected.
- Resetting configurations back to the default would correctly update the values in the Spinners.
- Saving changes and re-reading the Settings file would load the correct values in the Spinners.

4.2 Testing on a real device

Now that the Robotium testing was complete and the application was considered to be fit for purpose, some final testing was carried out on a real Android device. Time was devoted to this to ensure the following requirements were met:

- Text and icon size was big enough and clear.
- All buttons were easy to use.
- Tasks that the user carried out were completed with efficiency (e.g. populating the database did not take too long).
- Overall design of the application was attractive.

4.2.1 What was changed

A few design changes were made to meet the requirements declared in 4.2. The main addition was to add the feature that allows the user to swipe the screen to change the page of the script they are on. This was introduced as it was observed that having to press and hold a button each time the user wished to switched page was irritating. To do this, a gesture detector class was implemented which observed the user's finger movement on the screen. Depending on the direction and the speed of the finger's movement, the page will switch accordingly. The code can be found at Appendix G.

Another big change was to how a user records their dialogue. As previously described, when the user presses the record button, a dialog appears where the user can choose to begin recording. Once the recording is completed, the user moves to a new dialog where they enter a name for the audio. The main problem with this was that the application would be focused on the dialog while the user was recording, and the script would only be partially visible behind it. This meant the user had to either know the line correctly by memory, or write it down somewhere to read from. To get around this, the first dialog was dropped. When the user presses the record button, their device would instantly begin recording. The text of the record button would change to red while the chronometer shown in the first dialog was moved to the Main Screen. The chronometer would only be visible while the user was recording. When the user presses the record button again, they would be brought to the dialog where they can save their new audio file. With this new design, the user can now freely scroll through a page and also switch pages, all while still recording.

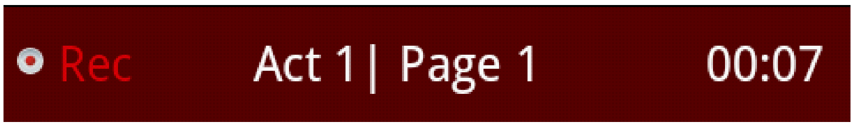
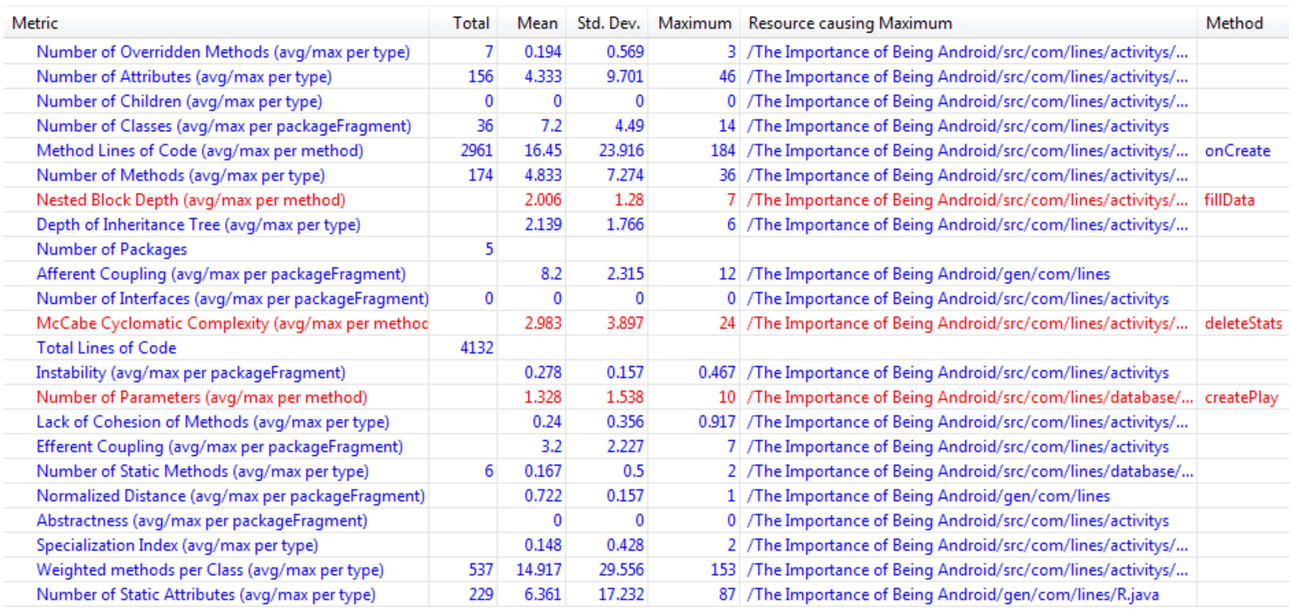


Figure 4.4: The application recording the user.
Located at the top of the Main Screen.

Finally several changes were made to improve the overall design of the application, such as icons, colours and text fonts. In particular, the text on the Main Screen was modified so it shared the same appearance as a traditional script. The character delivering the line was now entirely uppercase and the courier font was installed into the assets folder so it could be used.

4.3 Metrics Tool

Now that both implementation and testing was complete for the project, the final statistics of the metrics plugin were observed.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	7	0.194	0.569	3	/The Importance of Being Android/src/com/lines/activitys/...	
Number of Attributes (avg/max per type)	156	4.333	9.701	46	/The Importance of Being Android/src/com/lines/activitys/...	
Number of Children (avg/max per type)	0	0	0	0	/The Importance of Being Android/src/com/lines/activitys/...	
Number of Classes (avg/max per packageFragment)	36	7.2	4.49	14	/The Importance of Being Android/src/com/lines/activitys	
Method Lines of Code (avg/max per method)	2961	16.45	23.916	184	/The Importance of Being Android/src/com/lines/activitys/...	onCreate
Number of Methods (avg/max per type)	174	4.833	7.274	36	/The Importance of Being Android/src/com/lines/activitys/...	
Nested Block Depth (avg/max per method)		2.006	1.28	7	/The Importance of Being Android/src/com/lines/activitys/...	fillData
Depth of Inheritance Tree (avg/max per type)		2.139	1.766	6	/The Importance of Being Android/src/com/lines/activitys/...	
Number of Packages	5					
Afferent Coupling (avg/max per packageFragment)		8.2	2.315	12	/The Importance of Being Android/gen/com/lines	
Number of Interfaces (avg/max per packageFragment)	0	0	0	0	/The Importance of Being Android/src/com/lines/activitys	
McCabe Cyclomatic Complexity (avg/max per methoc		2.983	3.897	24	/The Importance of Being Android/src/com/lines/activitys/...	deleteStats
Total Lines of Code	4132					
Instability (avg/max per packageFragment)		0.278	0.157	0.467	/The Importance of Being Android/src/com/lines/activitys	
Number of Parameters (avg/max per method)		1.328	1.538	10	/The Importance of Being Android/src/com/lines/database/...	createPlay
Lack of Cohesion of Methods (avg/max per type)		0.24	0.356	0.917	/The Importance of Being Android/src/com/lines/activitys/...	
Efferent Coupling (avg/max per packageFragment)		3.2	2.227	7	/The Importance of Being Android/src/com/lines/activitys	
Number of Static Methods (avg/max per type)	6	0.167	0.5	2	/The Importance of Being Android/src/com/lines/database/...	
Normalized Distance (avg/max per packageFragment)		0.722	0.157	1	/The Importance of Being Android/gen/com/lines	
Abstractness (avg/max per packageFragment)		0	0	0	/The Importance of Being Android/src/com/lines/activitys	
Specialization Index (avg/max per type)		0.148	0.428	2	/The Importance of Being Android/src/com/lines/activitys/...	
Weighted methods per Class (avg/max per type)	537	14.917	29.556	153	/The Importance of Being Android/src/com/lines/activitys/...	
Number of Static Attributes (avg/max per type)	229	6.361	17.232	87	/The Importance of Being Android/gen/com/lines/R.java	

Figure 4.5: The final code metrics of the project.

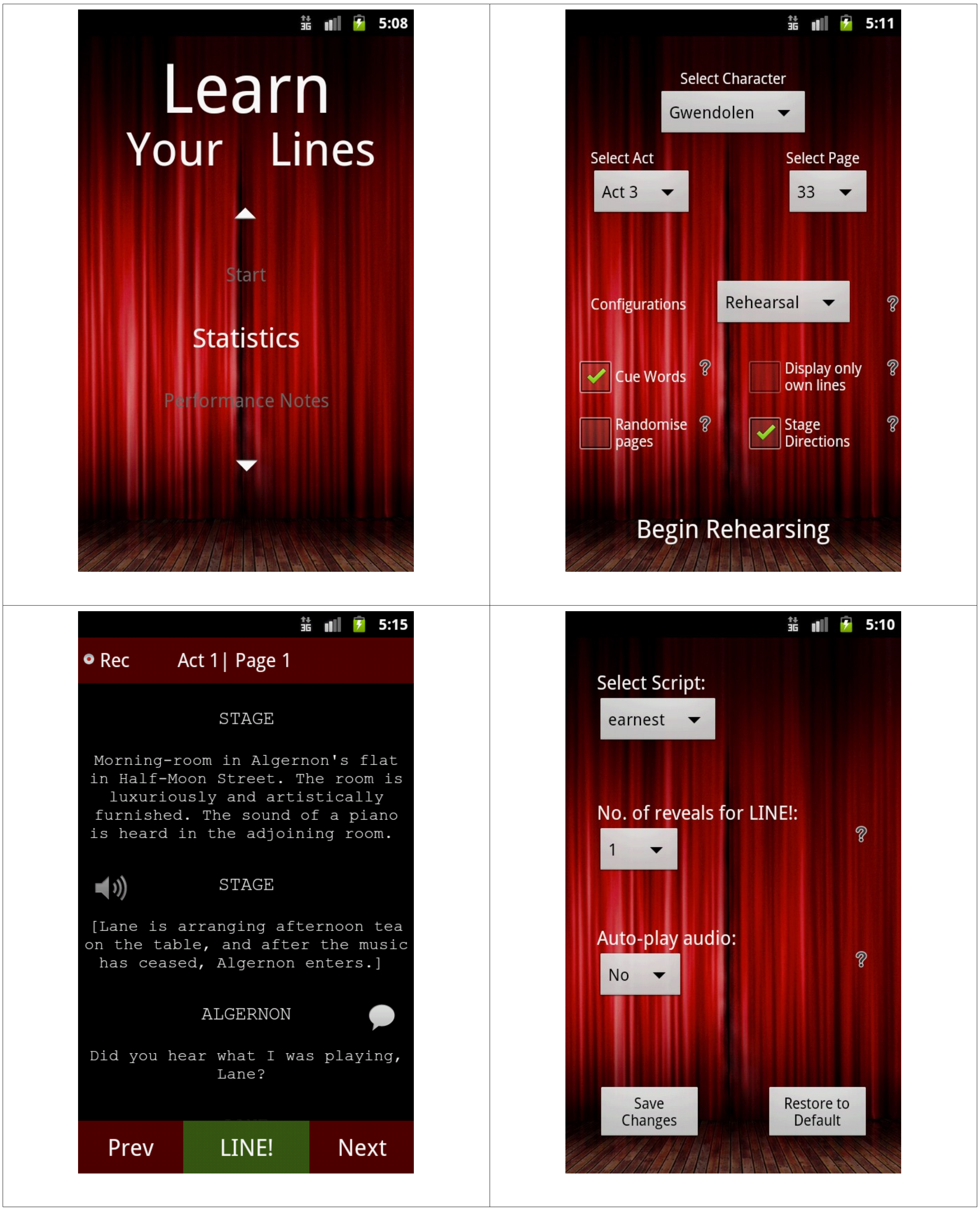
As mentioned in section 2.3, a metrics plugin for Eclipse was installed so good programming practice can be followed. As seen in Figure 4.5, three categories were highlighted as bad practice (given the maximum value):

- Nested Block Depth
- McCabe Cyclomatic Complexity
- Number of Parameters

The Nested Block Depth was directed at the method which populates the script, which has a depth of seven. This is due to the several different ways the script can be generated, based on the user's configurations. In brief, the method must iterate through all lines returned from a query, then check if we are in REHEARSAL mode, then examine if the current line is delivered by the user's character, then check if the user is revealing or hiding a line and finally check for the configurations the user has made e.g. cue words toggled on. Since this is the method that performs the most amount of work in the whole application, it is justified for having a large nested block depth.

Resetting stats for some selection was flagged for having a high cyclomatic complexity, with a value of 24. The McCabe Cyclomatic Complexity measures the number of linearly independent paths through a method. This method that has been highlighted does require multiple paths however as it first must determine which query to make to the database. Once the query has been made, we then have to iterate through each line and determine which stats need be reset i.e. values more then zero.

Finally the number of parameters for the method we use to create a new line in the database is flagged, with a value of ten. Again this can be justified as the number of parameters is given by the number of columns in the database, and cannot be avoided.


5:08

Learn Your Lines

Start

Statistics

Performance Notes

5:11

Select Character: Gwendolen

Select Act: Act 3

Select Page: 33

Configurations: Rehearsal

✓ Cue Words

Display only own lines

Randomise pages

✓ Stage Directions

Begin Rehearsing

5:15

Rec Act 1 | Page 1

STAGE

Morning-room in Algernon's flat in Half-Moon Street. The room is luxuriously and artistically furnished. The sound of a piano is heard in the adjoining room.

STAGE

[Lane is arranging afternoon tea on the table, and after the music has ceased, Algernon enters.]

ALGERNON

Did you hear what I was playing, Lane?

Prev | LINE! | Next

5:10

Select Script: earnest

No. of reveals for LINE!: 1

Auto-play audio: No

Save Changes

Restore to Default

Figure 4.6: Final state of app after testing was completed

Top: Home Screen, Options Screen
Bottom: Main Screen, Settings Screen

5 Evaluation

The final task that was carried out before the project was considered complete was to gather some feedback as part of an evaluative study.

5.1 Evaluation plan

The approach was to obtain feedback on the design of the application. Test users would interact with the app and give their opinions on how well it has been developed. Since evaluators are only commenting on the application's design, anyone could take part in the study, not just actors. The study involved completing several set tasks, and then answering a questionnaire.

5.1.1 User Tasks

Six tasks were selected for the user to complete. These tasks would allow the user to interact with the main features of the application, and require them to navigate through several screens. The evaluators were observed as they completed each task. It was noted which choices they made and which of the tasks gave them difficulty or they could not complete. If some time had passed and the user could not complete a task, they would be given a hint on what to do. The six tasks and the reasons for them are as follows:

- **Get the user to read a few pages of the script in NORMAL mode.** Here the user must first be able to interpret the Options Screen and make sure the configuration is set to NORMAL. Secondly, the user must know how to successfully "turn the page". It will be noted how they do this (i.e. press "Next" or swipe screen).

- **Interact with the script in REHEARSAL mode.** First the user is instructed to select a character and a page from which to begin. It is specified what selections the user must make. Next they are instructed to reveal three words from the currently hidden line. Once they have revealed those words, they must advance through the current page by revealing lines. It will be observed if the user can make a distinction between revealing words, and revealing the next line.

- **Have the user record themselves and apply the recording to a line.** First the user is told which line they need to record. Thus they must be able to navigate through the script and find the correct line. Next they must know how to begin recording and in turn save the recording when they are done. Finally, the user must apply the recording to the correct line and play it back. A few actions are observed on this task. First they must know how to bring up the "apply recording" option (press and hold on the relevant line). When they are taken to the Recordings Screen, they should be able to select the correct audio file. Finally they need to again select the correct line and play back the audio (again press and hold on the relevant line).

- **Create a Performance Note.** Here the user will perform many of the same actions as the last task so they should have learned from any mistakes they may have made previously. Once they have created the note, they must find it again from the Performance Notes screen then change its text. It will be noted which choice they make to navigate to the Performance Notes screen as there are many ways to do this.

- **View Statistics.** Again the user must make a choice as to how to view the Statistics screen. Once there, they must filter the stats so that they match the selections they made in the second task. Finally, they need to clear their selected stats.

- **Delete the recording and note which was created earlier.** For the final task, the user must again navigate to the different screens of the application to delete the recording and note they made. As well as observing how the user chooses to perform deletions, this also resets the app to its state prior to the user's interaction, thus ready for the next evaluator.

5.1.2 Questionnaire

Once the user had completed the tasks they were set, they had the chance to answer a questionnaire and give their opinions on the application. While the questionnaire itself would be anonymous, the user did have to identify themselves as one of the following:

- Experienced in using android smartphones.
- No experience in using android smartphones.

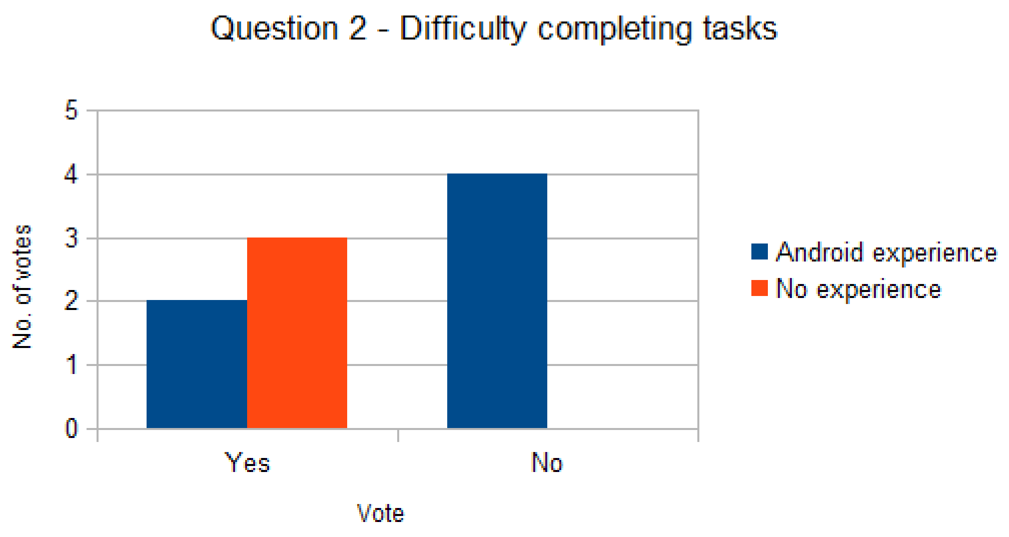
This way the results could be distinguished between users of different experience, and observed for any correlation between subsets of results. The user was then asked a further seven questions:

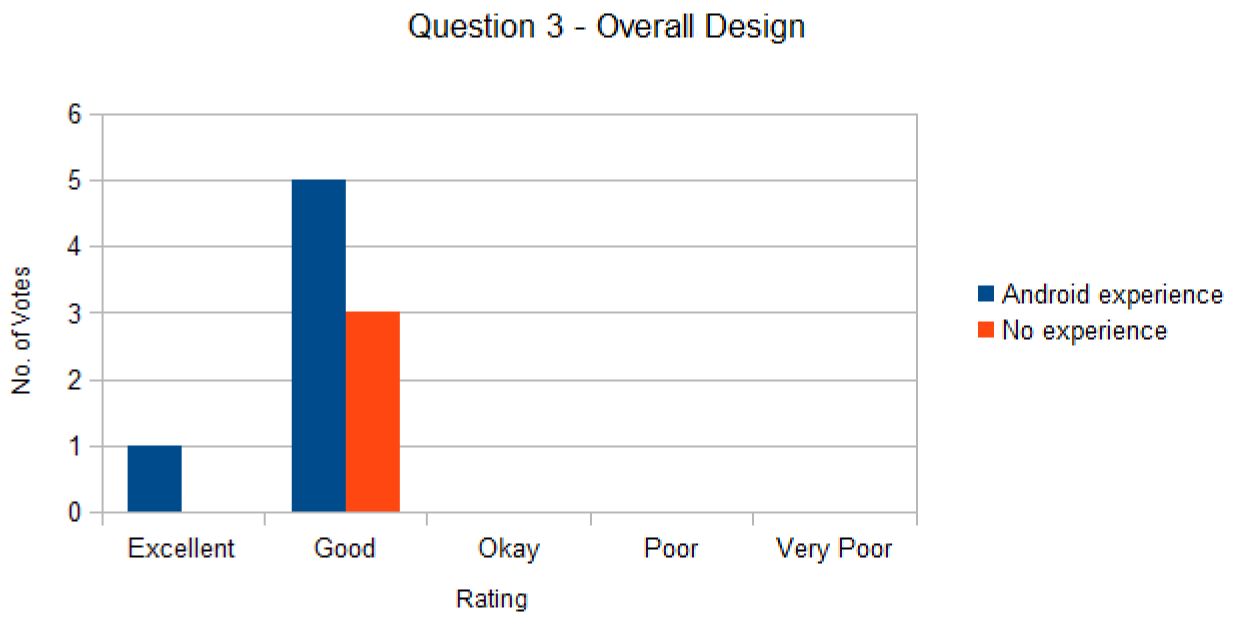
- “Did you have any difficulty in completing the tasks?”
- “How would you rate the overall design of the app?”
- “Was there any part of the design you liked in particular?”
- “Was there any part of the design you didn't like, or found confusing?”
- “Are there any features that weren't in the app that you feel should have been included?”
- “Finally, would you use the “Learn Your Lines” app in the future to rehearse?” (Respondents were asked this question in the assumption they were actors)

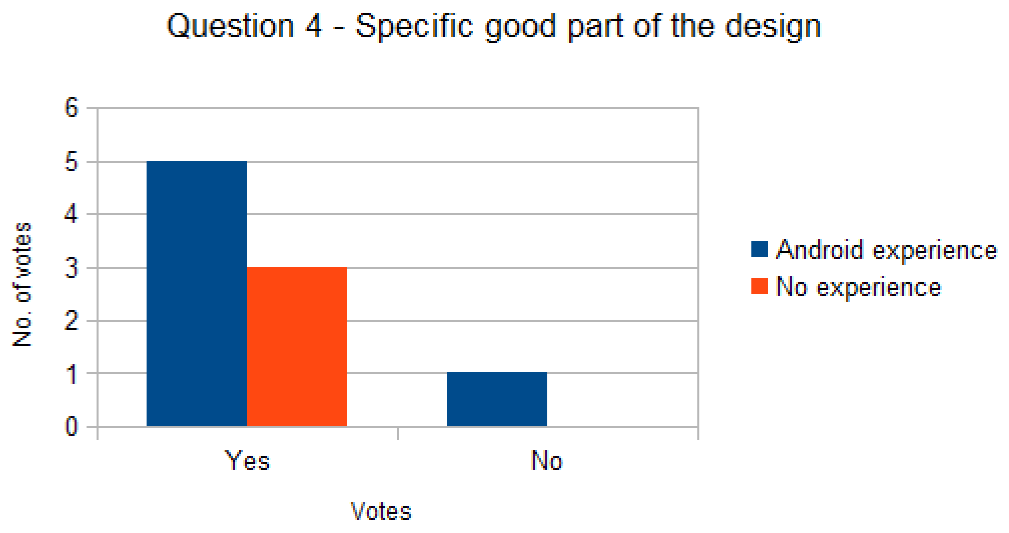
As well as the questions above, the user could give their opinion on each of the features they encountered while completing the tasks if they were useful in helping actors rehearse. By analysing the answers from the questionnaire, combined with observing the performance of task completion, we should be able to identify where users feel some improvements are required and also which parts have been developed well. Ultimately, we would hope to achieve confirmation that the requirements of the project have been met, and has been completed successfully. A sample sheet of both the user tasks and the questionnaire can be found at Appendix H and I respectively.

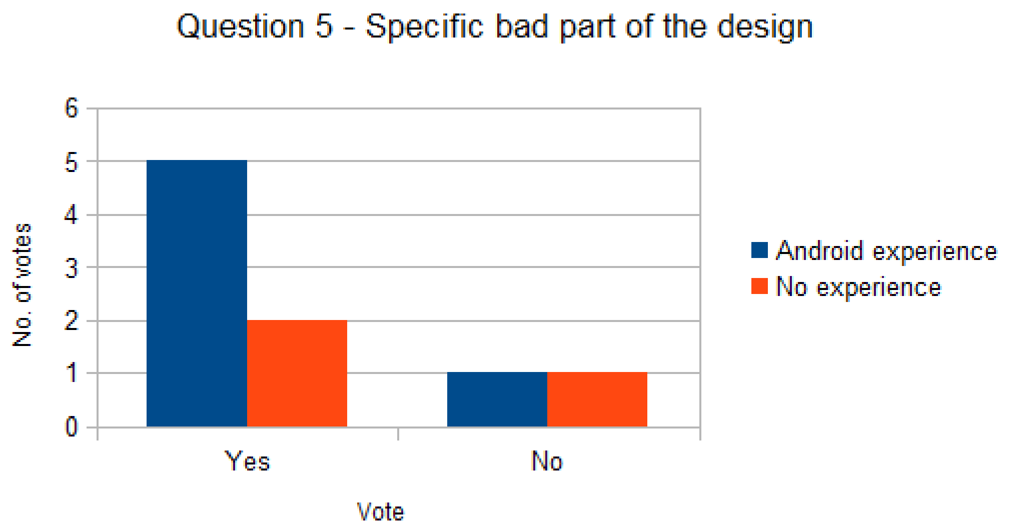
5.2 Evaluation Results

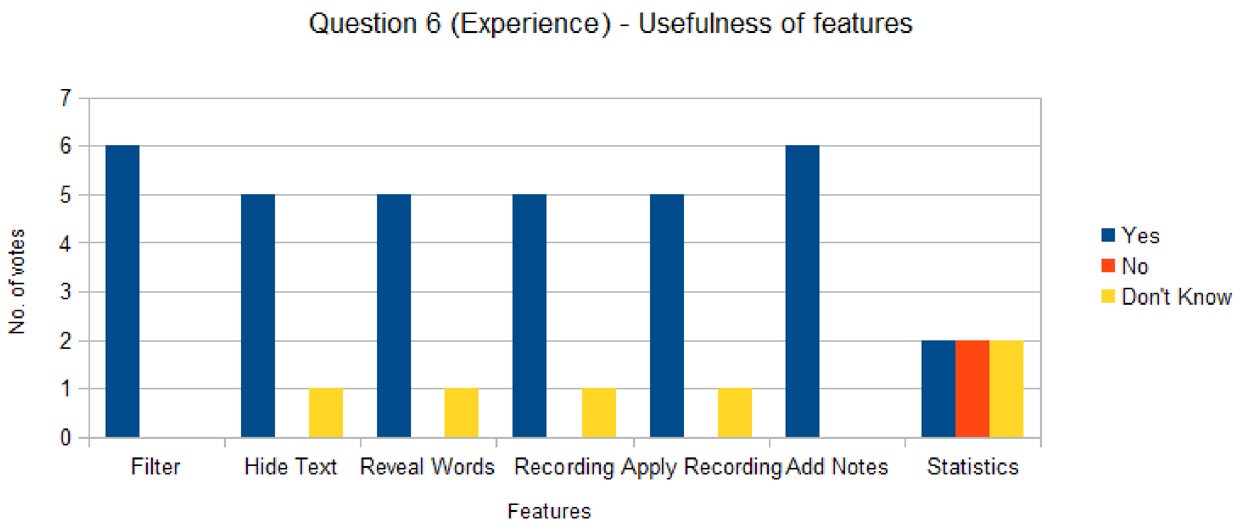
In total, nine evaluators took part in the study; six who had experience using an Android device, and three who had no experience. Unfortunately none of respondents had any experience in acting. However, the results obtained still proved to be valuable to the project.

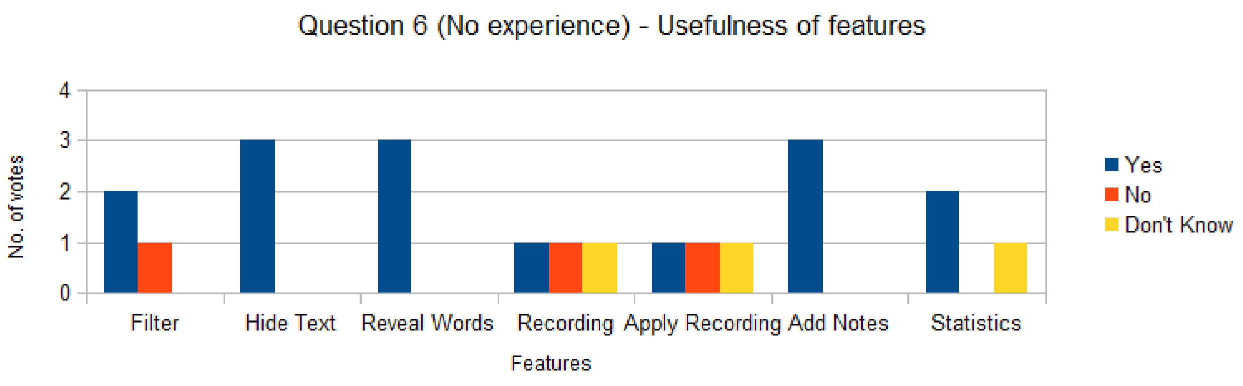


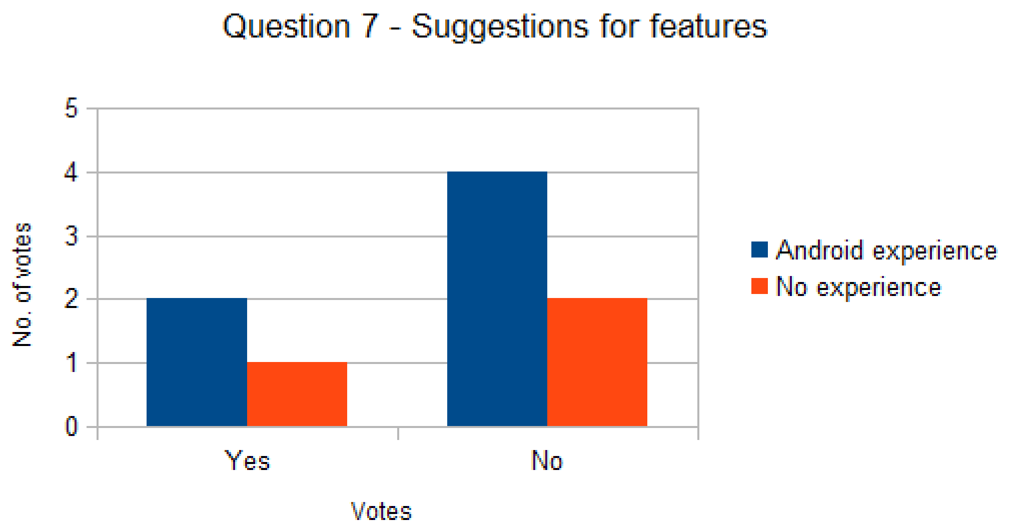



Question 4 - Specific good part of the design
No. of votes
0
1
2
3
4
5
6
Yes
No
Votes
Android experience
No experience


Question 5 - Specific bad part of the design
No. of votes
0
1
2
3
4
5
6
Yes
No
Vote
Android experience
No experience


Question 6 (Experience) - Usefulness of features
No. of votes
0
1
2
3
4
5
6
7
Filter
Hide Text
Reveal Words
Recording
Apply Recording
Add Notes
Statistics
Features
Yes
No
Don't Know





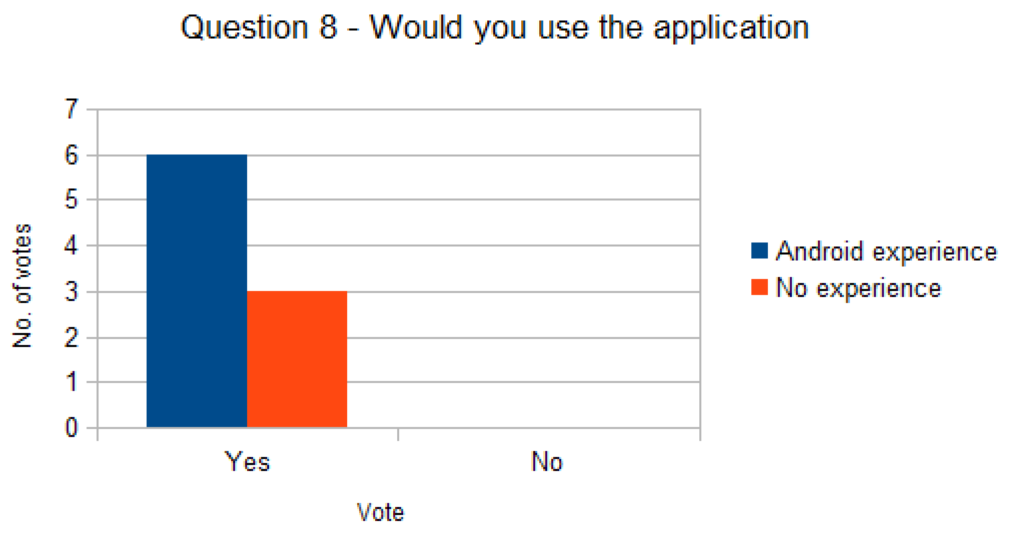


Figure 5.1: The results for each question in the questionnaire

5.3 Analysis of Results

Overall, feedback for the project was positive. Users of both categories indicated that the overall design was mostly good. Many compliments were aimed at the recording and notes feature, and also that the design was attractive and user-friendly. The design was not without criticism however, as respondents cited the Options screen as being a little cluttered. Of all the features that user's interacted with when completing the tasks, most of them were described as useful, with mixed opinions for the statistics feature.

Users were able to complete tasks mostly with success. As expected, respondents that had no prior Android experience did have more trouble but, nonetheless, still managed to complete the tasks. Users did consistently have trouble when instructed to advance through a page in REHEARSAL mode and apply a recording to a line. When trying to advance to the next line, users would repeatedly press the prompt button and reveal one word at a time. This was due to the “Next” button having two functions, and users thought it would only move to the next page. When trying to apply a recording, evaluators did not know they had to press and hold on a line to bring up the context menu. However apart from these two problems, overall the application was easy to use and the evaluative study confirmed that the app was suitable for users of different Android experience.

6 Further Work and Conclusion

6.1 Suggestions for Further Work

Currently the main functionality of the application is to hide the user's character's lines while they rehearse. Once the user believes their recollection to be correct, they advance through the page to their next line. The point of this is to discourage the user from “peeking” at their line when they forget what it should be.

An interesting next step for this feature is to record the user delivering their line and let the device interpret the audio and check that the user was correct. Although speech-to-text technology currently exists for Android devices, it is designed to replace typing. Thus users need to speak clearly and slowly in order for the device to accurately interpret the speech. Obviously this is not suitable for an actor trying to perform in character. Devoting time to research into other engines that perform better or developing the technology from scratch would be a valuable step forward for the project.

An extension on this would be to also develop text-to-speech. Instead of having the user create recordings and apply the audio to lines manually, their device could simply “read out” the other character's lines. While this would be a useful feature, there are reasons against it. If the actor is rehearsing and trying to stay in character, it can be distracting if the other character they are rehearsing with is performed by an automated voice that could easily mispronounce words that it does not recognise. However, it would be useful to give the actor both options, so they can choose between an automated voice or an audio of their choosing.

6.2 Conclusion

The purpose of this project was to develop an Android application that helped actors rehearse for a role in a play. To help the users learn the lines of their script, the app would hide their selected character's lines until they chose to proceed. As well as being a tool to aid the memorisation of the text, the application was designed to be a “Digital Partner”. This meant that and actor could treat their device as another person; taking on the roles of the other characters in the script and providing feedback through recordings and statistics.

The first set of tasks for the project was to perform some background reading on how actors rehearse for their roles, looking for popular techniques and methods used that could be replicated in the application. As well as this, time was spent looking into other apps that provided similar functionality as was intended for this project. While a few successful apps existed for Apple devices, their was a limited number available for Android. Of those available, they were not implemented particularly well. This confirmed that this project could add something new to the market.

Most of the project time was devoted to implementing the application. Effectively, the application could load scripts and allow the user to rehearse from their script. Additional features were then added to aid the rehearsing experience, such as adding performance notes and creating recordings. Some further time was spent exhaustively testing the application using both the Android Testing Framework Robotium, and a real Android device. The design was improved at this stage as well as adding some minor features to increase the ease of use.

Finally the project was evaluated by potential users of the application. Respondents had to complete a range of set tasks that would observe the quality of the app's design, and then fill out a questionnaire that would allow them to provide their opinion. Overall, the app was well received, with some minor problems identified.

Finally, the main goals of the project that were set out have all been met. As well as the functional requirements that were listed, the non-functional requirements have been met too; delivering an application that is easy to use and is robust. With that said, it can be concluded that this project has been a success.

Appendix A Parsing the text file

```
// Read line-by-line and pick out the relevant information
while (dis.available() != 0) {
        lineNo++;

        line = dis.readLine();

        // Split line into array of words
        String words[] = line.split("\\s+");
        String firstWord = words[0];

        // Keep a count of which Act we're on
        if (words[words.length - 1].equals("ACT")) {
                actNo++;
        }

        // Keep count of what page we're on (23 lines/page)
        if ((lineNo % 23) == 0 && lineNo != 0) {
                pageNo++;
        }

        // Check our firstWord is a character name
        if (isCharacter(firstWord)) {
                // If the first word doesn't contain a period (.) then we need
                // to take the second word as well as part of the character
                // name.
                if (!firstWord.contains(".")) {
                        firstWord += " " + words[1];

                        text = "";
                        for (int j = 2; j < words.length; j++) {
                                text += words[j] + " ";
                        }
                        // If the second word is "and" then it is two characters
                        // delivering a line and we need to get the other
                        // characters name.
                        if (words[1].equals("and")) {
                                firstWord += " " + words[2] + ".";

                                text = "";
                                for (int j = 3; j < words.length; j++) {
                                        text += words[j] + " ";
                                }
                                // Handle the rest of the data that hasn't yet been
                                // filtered
                        } else if (!words[1].contains(".")) {
                                firstWord = "STAGE.";

                                text = "";
                                for (int j = 0; j < words.length; j++) {
                                        text += words[j] + " ";
                                }
                        }
                }
        // If the firstWord isn't a character, then it is a stage
        // direction
        } else {
                firstWord = "STAGE.";

                text = "";
                for (int j = 0; j < words.length; j++) {
                        text += words[j] + " ";
                }
        }

        // If we didn't manage to populate "text" from the previous if
        // statements, then do it here.
        if (text.equals("")) {
                for (int j = 1; j < words.length; j++) {
                        text += words[j] + " ";
                }
        }

        // Once we have all the data picked out from current line in text
        // file, create a new row in the database. Filter out text we don't
        // want in our database (Key words).
        if (!isAllUpperCase(words[0])) {
                firstWord = firstWord.substring(0, firstWord.length() - 1);
                mDbAdapter.createPlay(lineNo, firstWord, text, actNo, pageNo,
                                "N", "N", 0, 0, 0);
                // If we're not adding to the database, then we need to reduce
                // the line count.
        } else {
                lineNo--;
        }

        // Clear "text" before we read next line
        text = "";
}
```

Appendix A: Parsing the text file that stores the script and loading into the database

Appendix B Populating the Spinners

```java
private void populateCharacters() {
	mCursor = mDbAdapter.fetchAllLines();

	// First get the data from "character" column and filter out unwanted
	// characters (e.g. STAGE)
	if (mCursor.moveToFirst()) {
		do {
			String character = mCursor.getString(mCursor
					.getColumnIndex("character"));
			if (!(character.equals("STAGE") || character.contains("and"))) {
				characters.add(character);
			}
		} while (mCursor.moveToNext());
	}

	HashMap<String, Integer> charOccur = new HashMap<String, Integer>();
	// Get the number of lines spoken by each character and store in HashMap
	Set<String> unique = new HashSet<String>(characters);
	for (String key : unique) {
		charOccur.put(key, Collections.frequency(characters, key));
	}

	characters.clear();

	// Sort character list based on the number of lines they have
	while (charOccur.size() > 0) {
		int max = Collections.max(charOccur.values());
		characters.add(getKeyByValue(charOccur, max));
		charOccur.remove(getKeyByValue(charOccur, max));
	}

	// Set contents of Character Spinner
	mAdapterChar = new ArrayAdapter<String>(OptionsActivity.this,
			android.R.layout.simple_spinner_item, characters);
	mAdapterChar
		.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
	mChar.setAdapter(mAdapterChar);

	mCursor.close();
}
```

Appendix B: Dynamically populating the character Spinner

Appendix C Displaying the Script

```
if (rehearsal) {
	if (currentChar.equals(character)) {
		// If we are revealing the next line, then we need to
		// keep a count of when to stop
		if (command.equals("forward") && visibleLines >= 1) {
			newLine = mCursor.getString(mCursor.getColumnIndex("line"));
			visibleLines--;
		// Similar for hiding current line, we need to keep
		// a count of when to stop
		} else if (command.equals("back") && visibleLines >= 3) {
			newLine = mCursor.getString(mCursor.getColumnIndex("line"));
			visibleLines--;
		} else {
			// Before we exit, store the current line
			currentLine = mCursor.getString(mCursor.getColumnIndex("line"));
			mCursor.moveToLast();
		}
	} else {
		// If we haven't reached the user's current line, then
		// store it for showing
		newLine = mCursor.getString(mCursor.getColumnIndex("line"));
	}
}
```

Appendix C: Populating the list of lines
when the user is in rehearsal mode.

Appendix D   Revealing words

```java
// Only obtain next word if there are any left
if (visibleWords <= words.length) {
	String line = "";
	// Construct the new line revealing the correct amount of words
	for (int i = 0; i < visibleWords; i++) {
		line += words[i] + " ";
	}
	// Update the line we are working with
	lines.remove(lines.size() - 1);
	Line newLine = new Line(character, line);
	lines.add(newLine);

	// Update the Listview
	LineAdapter adapter = new LineAdapter(this, R.layout.script_list_layout, lines);
	setListAdapter(adapter);

	this.setSelection(adapter.getCount());

	visibleWords++;
}
```

Appendix D: Revealing words from a hidden line

Appendix E Displaying statistics

```
private void showStats() {
        String character = mChar.getSelectedItem().toString();
        String act = mAct.getSelectedItem().toString();
        String page = mPage.getSelectedItem().toString();
        float views = 0;
        float prompts = 0;
        float completions = 0;

        getTotalStats();

        mCursor = mDbAdapter.fetchCharacter(character, page);
        String words[] = act.split("\\s+");

        // If all spinners have a specific item selected
        if (!character.equals("All") && !act.equals("All")
                        && !page.equals("All")) {
                mCursor = mDbAdapter.fetchCharacter(character, page);
                // If only the character spinner is selected as "All"
        } else if (character.equals("All") && !page.equals("All")) {
                mCursor = mDbAdapter.fetchPage(page);
                // If only the page spinner is selected as "All"
        } else if (!character.equals("All") && !act.equals("All")
                        && page.equals("All")) {
                mCursor = mDbAdapter.fetchFilteredPages(words[1], character);
                // If both the act and page spinner is selected as "All"
        } else if (!character.equals("All") && act.equals("All")
                        && page.equals("All")) {
                mCursor = mDbAdapter.fetchAllFilteredPages(character);
                // If both the character and page spinner is selected as "All"
        } else if (character.equals("All") && !act.equals("All")
                        && page.equals("All")) {
                mCursor = mDbAdapter.fetchAllPages(words[1]);
        }

        // Get the stats for the current selection
        if (mCursor.moveToFirst()) {
                do {
                        views += mCursor.getInt(mCursor.getColumnIndex("views"));
                        prompts += mCursor.getInt(mCursor.getColumnIndex("prompts"));
                        completions += mCursor.getInt(mCursor
                                        .getColumnIndex("completions"));
                } while (mCursor.moveToNext());
        }

        // Finally, if all spinners are selected as "All"
        if (character.equals("All") && act.equals("All") && page.equals("All")) {
                views = totalViews;
                prompts = totalPrompts;
                completions = totalCompletions;
        }

        // Avoid divide by zero error
        if (totalViews == 0) {
                totalViews++;
        }
        if (totalPrompts == 0) {
                totalPrompts++;
        }
        if (totalCompletions == 0) {
                totalCompletions++;
        }

        // Calculate percentages for each statistic
        float viewsPercent = (views / totalViews) * 100;
        float promptsPercent = (prompts / totalPrompts) * 100;
        float completionsPercent = (completions / totalCompletions) * 100;

        // Format percentages
        String viewsFormat = String.format("%.0f", views);
        String promptsFormat = String.format("%.0f", prompts);
        String completionsFormat = String.format("%.0f", completions);
        String viewsPercentFormat = String.format("%.1f", viewsPercent) + "%";
        String promptsPercentFormat = String.format("%.1f", promptsPercent)
                        + "%";
        String completionsPercentFormat = String.format("%.1f",
                        completionsPercent) + "%";
                mViewsNum.setText(viewsFormat);
        mViewsPercent.setText(viewsPercentFormat);
        mPromptsNum.setText(promptsFormat);
        mPromptsPercent.setText(promptsPercentFormat);
        mCompleteNum.setText(completionsFormat);
        mCompletePercent.setText(completionsPercentFormat);
}
```

Appendix E: Displaying statistics to the user
based on their selection

Appendix F   Auto-playback Audio

```
private void findNextAudio() {
	for (int i = playbackPosition; i < lines.size(); i++) {
		Cursor line = mDbAdapter.fetchLine(getLineNumber(i));
		String audio = line.getString(line.getColumnIndex("audio"));

		if (hiddenLineNo == getLineNumber(i)) {
			break;
		}

		// If we find a line with an audio file, then update the latest
		// playbackPosition, and play the audio file
		if (!audio.equals("N")) {
			try {
				playbackPosition = i;
				playSelectedRecording(playbackPosition, true);
				playbackPosition++;
				break;
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}
}

private void playSelectedRecording(final long id, final boolean autoplay)
		throws Exception {
	if (audioApplied(id)) {
		// If the selected line is hidden, then we don't want to play the
		// audio until the line is revealed
		if (rehearsal && (hiddenLineNo == getLineNumber(id))) {
			Toast.makeText(getApplicationContext(),
			"Cannot play recording while line is hidden. No cheating!",
			Toast.LENGTH_LONG).show();
		} else {
			Cursor line = mDbAdapter.fetchLine(getLineNumber(id));
			String filename = line.getString(line.getColumnIndex("audio"));

			mStopPlayBack.setEnabled(true);
			mStopPlayBack.setVisibility(View.VISIBLE);

			ditchPlayer();
			player = new MediaPlayer();
			player.setDataSource(filename);
			player.prepare();
			player.start();
```

```
                        // When the audio file has stopped playing back, then we want to
                        // find the next audio file that we want to play
                        player.setOnCompletionListener(new OnCompletionListener() {
                              public void onCompletion(MediaPlayer mp) {
                                    mStopPlayBack.setEnabled(false);
                                    mStopPlayBack.setVisibility(View.INVISIBLE);
                                    if (autoplay) {
                                          findNextAudio();
                                    }
                              }
                        });
                  }
            }
      }
```

Appendix F: Recursive algorithm to auto-playback audio.
“findNextAudio” calls “playSelectedRecording”
which then calls back to “findNextAudio”

Appendix G   Gesture Dectector Class

```
class MyGestureDetector extends SimpleOnGestureListener {
	@Override
	public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
				float velocityY) {
		try {
			if (Math.abs(e1.getY() - e2.getY()) > SWIPE_MAX_OFF_PATH)
				return false;
			// right to left swipe
			if (e1.getX() - e2.getX() > SWIPE_MIN_DISTANCE
					&& Math.abs(velocityX) > SWIPE_THRESHOLD_VELOCITY) {
				viewFlipper.setInAnimation(slideLeftIn);
				viewFlipper.setOutAnimation(slideLeftOut);
				switchPage(true);
			} else if (e2.getX() - e1.getX() > SWIPE_MIN_DISTANCE
					&& Math.abs(velocityX) > SWIPE_THRESHOLD_VELOCITY) {
				viewFlipper.setInAnimation(slideRightIn);
				viewFlipper.setOutAnimation(slideRightOut);
				switchPage(false);
			}
		} catch (Exception e) {
			// nothing
		}
		return false;
	}
}
```

Appendix G: Gesture Detector class that switches page accordingly based on user's movement

Appendix H User Tasks

The Importance of Being Android
Evaluative Study

Your task is to be part of an evaluative study into the design of a smartphone application. The app in question is "Learn Your Lines", the purpose of which is to help actors and actresses prepare for a role in a play. Although it is aimed for those working in the theatre, it can be extended to any medium that involves learning, memorising and rehearsing from a script. The app will be using the screenplay for “The Importance of Being Earnest” by Oscar Wilde, for this evaluation.

You will be asked to complete a series of tasks, and your performance in each task will be assessed so the design of the application can be evaluated. All the tasks should be completed within ten minutes and then it will be required that you complete a short questionnaire. This whole process will be completely anonymous. Do not hesitate to ask if you have any questions.

NOTE: On the other side of this page you will find a diagram which shows what the buttons do on the device that is provided to you.

Task 1 – Viewing the Script

- View the script in “Normal” mode. Begin from the first page. When the script appears, advance through it until you reach page five. Reading the script is optional.

Task 2 – Viewing the Script in Rehearsal mode

- Return to the Options screen and switch from “Normal” mode to “Rehearsal” mode. Select “Algernon” as your character, and begin from Act 2, page 23. Now select "Begin Rehearsing".

- Reveal three words from the currently hidden line. After that, advance through the page until you reach Algernon's fourth line. (The previous line will be delivered by Jack and will contain the text; “Well, at any rate, that is better...”).

Task 3 – Record yourself rehearsing a line

- Record yourself speaking Algernon's third line on page 23. ("Yes, if you are not too long...).

- Save your Recording as “Algernon”. You may press “Cancel” before saving, and record yourself again if you wish.

- Now apply the recording you just made to the relevant line.
- Finally playback the recording you just applied to the line.

Task 4 – Create a note

- Create a note for the same line you just created a recording for. You may enter any text of your choosing.
- Go to the Performance Notes screen and find the note you just created. Rename the title of the note to; “Algernon – Line 6”.

Task 5 – View your Stats

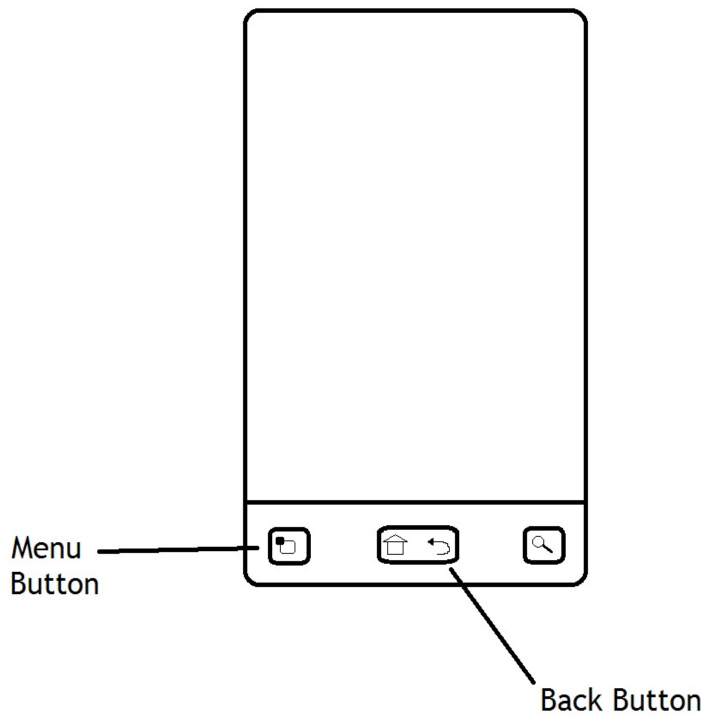
- Now go to the Statistics screen. Filter the stats so the selected character is “Algernon” and view for all pages on Act 2.
- Clear the stats for the current selection.

Task 6 – Deletion

- Go to the Recordings screen and delete the recording you made earlier.
- Finally, go to the Performance Notes screen and delete the note you made earlier.

Button Help

Below is a diagram with the buttons available on the phone. The only buttons you should need to use is the menu button and the back button.



Appendix H: Task Sheet the user is presented with for evaluative study

Appendix I Questionnaire

The Importance of Being Android Questionnaire

Question 1:

Select the most appropriate (Please only tick one).

☐ Experienced in using android smartphones

☐ No experience in using android smartphones

Question 2:

Did you have any difficulty in completing the tasks?

☐ Yes ☐ No

If Yes, please specify the tasks you had problems with, and why.

Question 3:

How would you rate the overall design of the app?

☐ Excellent ☐ Good ☐ Okay ☐ Poor ☐ Very Poor

Question 4:

Was there any part of the design you liked in particular?

☐ Yes ☐ No

If Yes, please specify the parts you liked and why

Question 5:

Was there any part of the design you didn't like, or found confusing?

☐ Yes ☐ No

If Yes, please specify the parts you didn't like, or found confusing, and why

Question 6:

For each of the following features you used in the app, would you describe them as useful tools in improving the rehearsing experience for acting?

- **Filtering specific pages:**

☐ Yes ☐ No ☐ Don't Know

- **Hiding text until proceeding through script:**

  ☐ Yes ☐ No ☐ Don't Know

- **Revealing words from hidden text:**

  ☐ Yes ☐ No ☐ Don't Know

- **Recording audio:**

  ☐ Yes ☐ No ☐ Don't Know

- **Applying recordings to specific lines:**

  ☐ Yes ☐ No ☐ Don't Know

- **Adding notes to lines:**

  ☐ Yes ☐ No ☐ Don't Know

- **Record statistics as you rehearse script:**

  ☐ Yes ☐ No ☐ Don't Know

Question 7:

Are there any features that weren't in the app that you feel should have been included?

☐ Yes ☐ No

If Yes, please specify the features you would have liked included

Question 8:

Finally, would you use the “Learn Your Lines” app in the future to rehearse?

☐ Yes ☐ No

Appendix I: Questionnaire for evaluation

Bibliography

[1] O. Wilde. *The Importance of Being Earnest*.
http://www.gutenberg.org/files/844/844-h/844-h.htm

[2] *Robotium Toolkit*.
http://code.google.com/p/robotium/

[3] G. Guberek. *Acting: How to Learn Lines*.
http://www.helium.com/items/822355-acting-how-to-learn-lines

[4] *Rehearsal*.
http://www.rehearsaltheapp.com/

[5] *Rehearsal Assistant/Voice Recrd*.
https://play.google.com/store/apps/details?id=urbanstew.RehearsalAssistant&hl=en

[6] *Line Please – Acting Tool*.
http://lineplea.se/

[7] *Metrics 1.3.6*.
http://metrics.sourceforge.net/

[8] *Dashboards | Android Developers*.
http://developer.android.com/about/dashboards/index.html

[9] *Stackoverflow.*
http://stackoverflow.com/questions/6356170/how-should-i-open-and-close-my-database-properly

[10] *Android – Recording Audio*.
http://www.youtube.com/watch?v=XANjoeEeQ1Y

[11] *Stackoverflow.*
http://stackoverflow.com/questions/14219084/comparing-two-strings-in-java-and-identify-duplicate-words

[12]*McCabe Cyclomatic Complexity - Insight 9.6*.
http://www.klocwork.com/products/documentation/current/McCabe_Cyclomatic_Complexity