**Table of Contents  
  
Abstract**

1. **Introduction**
   1. Current existing technologies
   2. Analysis of previous work
   3. Problem definition and scope
   4. Software Engineering analysis
2. **Description of Hardware and Software**2.1 Hardware  
   2.2 Software
3. **Theoretical Tools- Analysis and Development**
   1. Use Case Dig.
   2. Class Dig.
   3. Sequence Dig.
   4. Activity Dig.
   5. Component Dig.
   6. Flow Dig.
4. **Development of Software**4.1 Design Phase  
   4.2 Implementation Phase
5. **Testing and Analysis**5.1 Testing  
   5.2 Analysis
6. **Activity Time Chart**
7. **Conclusions**
8. **Recommendations and Future Work**  
    **References**

**Abstract**

If God has not given some capability, it’s the reason only almighty knows. It’s a bitter truth that India is boasting the highest percentage of world’s blind population. It is well known that power of speech has its own significance and unless it is fixed in the form of some written expression or text, the weight of speech is very light. It forms the soul of speech. So, in this semester we have tried to contribute to our society with an effort to make it easier for visually impaired to fix their thoughts. We have chosen Android for it is now widely available in gadgets. Arguably, someone would say that what made us to do that? And most obvious answer is the cost involved in purchasing a physical Braille Typewriter. As tablets are now available in around 2k rupees its lot cheaper. So using such a simulator which can simulate text on mobile/tablet through touch would surely enhance portability and ease of use (as now we can get rid of those bulky machines).

**1.Introduction**

1.1 Current Existing Technologies:

Presently there are some simulators which provide braille like interface primarily for English. Most practical of all is a Stanford Project [1]. They have created it for the same reason as we have in mind, Portability, cost and usability. But there are also some apps like this [2] and this [3] which merely treats it as a kind of some game but these do not let disabled to actually type and save the text.



(Figure showing a standard Braille typewriter)

Bulky Machines (as depicted by the figure) are very inconvenient to carry. Also proper sitting position is required to type on them. But we do agree to the fact that they readily produce printed documents. But we can’t listen to them or put them in digitized form as we would be able to do through our application.

1.2 Analysis of previous work in this area

Braille script was originally invented by Louis Braille in 1824 when he was merely 15 years old and published in 1829. He made uniform columns of six dots each with some dots raised (embossed).Braille script works on the combination of six dots arranged in a specific pattern. Braille’s cells were capable of being recognized as letters with a single touch of finger [4].

C:\Users\Bond\Desktop\cell.pngSample of one braille Cell –

Hindi Braille is an adaptation of standard Louis Braille script. It basically works on phonetics. Phonetics is a branch of linguistics that comprises the study of the sounds of human speech or in case of sign languages the equivalent aspects of sign. Hindi Braille assigns the cells to the basic sounds of the Indian languages (these are called aksharas) in a manner where vowels and consonants that find direct equivalents in English are given the same representation as in English. This way, with minimal effort one would be able to read both English text and Hindi language text. This arrangement is essential if the visually handicapped are required to communicate with their counterparts in other countries.

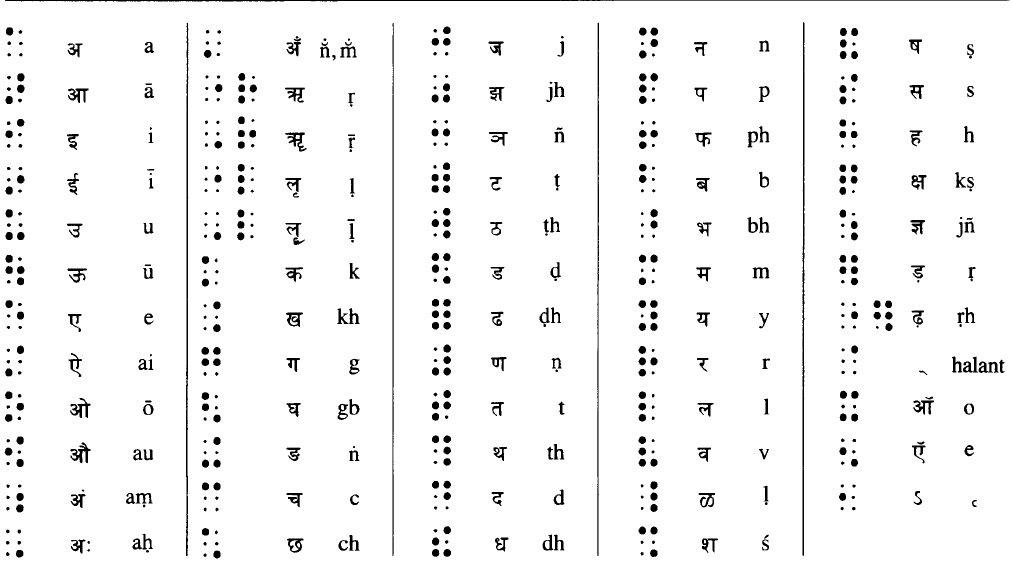
Also, Hindi Braille retains all the basic conventions relating to the representation of numerals, punctuation and special symbols just as in Standard English Braille. There are no concepts in Hindi Braille like the lines in English Braille. One has to necessarily learn the cell assignments for the different vowels and consonants and hence the initial period of training has to be prolonged.

1.3 Problem Definition and Scope

Our target is to develop an application which facilitates visually impaired users to type in Hindi Braille with advantages like portability and cheaper cost over traditional typewriters.

We are using Indian Standard Hindi (**Devanagari)** Braille Script accepted by UNESCO [5][6].

Following is the mapping used in the development:



(Taken from UNESCO world Braille usage [5])

Each of the above combinations is mapped to corresponding **UNICODE** values.

Following is the mapping used for the development of English braille:



(Figure showing English mappings from Wikipedia [4])

Our application supports *addition of multiple languages* (as long as supported by the Android OS itself) with just an addition of two text(.txt) files, one containing the characters and the other containing the corresponding mapping values for the respective characters. User would be able to save file and in turn can “listen” to the previously saved files using simple backward and forward gestures.

It provides the facility to the user to listen what he/she has written. The saved files would reside in the file system of android and can be used to take print-out from braille printer or can be directly used for normal printer so that other can also read their magnificent thoughts.

1.4 Formulation of Present Problem

i. Bulkiness –

Bulky typewriters are to be replaced with sleek, handy and portable tablets which are far more easy to use and handle.

ii. Cost –

Purchase and maintenance cost are significantly lower than that of traditional braille writer.

iii. Accessibility –

Instead of simple paper embossed characters, audio feedback is provided with features like deletion and reading of saved files with visual representation to facilitate the case if someone wants to read or would like to take a print out.

1.5 Organization of the Thesis (Software Engineering Analysis) [7]

The main phases involved in software engineering are:  
 a. System Identification

b. System Modeling

c. System Integration

d. System Installation

Modeling presents an abstract view, Integration means putting hardware, software and people together to make a system and In Installation system has to be installed in the customer’s environment.

For our target application, we would be using **Component Based Software Engineering.** In CBSE various components are integrated to form a system. This strategy encourages reuse and also exploits new component standards.

1. Component Analysis:   
     
   There are five main Components in our application -  
    i. Text Area (Display Area)  
   ii. User Interaction Area  
   iii. Processing Interface  
   iv. File Handler  
   v. Shake Handler
2. Requirement Modification:

If more combinations introduced in the future, the Processing interface could be modified without affecting the whole application framework.

c. System Design with reuse:

Shake handler and File handler are two independent components of the application. Can be used with any other component seamlessly for handling shake and file IO events. Some features of dynamic button handler can also be used as components.

d. Development and Integration:

Each component developed independently and integrated into the system (except for the core handling routines).

**2. Hardware and Software Requirements:**

2.1 Hardware

For smooth performance and seamless integration into the database and touch interface, min 1Gz dual core processor with 512MB of RAM is required. As the application is largely dependent upon the multi-touch capabilities of the touch screen, the screen should be capable of identifying at least *eight* simultaneous touch events i.e. minimum eight simultaneous finger touch at a time.

2.2 Software

The latest version of Android available would be most suitable for proper functioning. But still minimum version supported is Android 2.3.6 (Gingerbread) with gesture detection enabled. Also user would have to give external storage access permission to the application along with vibrator access to experience a better application output.

**3. Theoretical Analysis:**

3.1 Use Case Diagram

Use Case Diagrams describe the functionality of a system and users of the system. These diagrams contain the following elements:

**Actors**, which represent users of a system, including human users and other systems.

**Use Cases**, which represent functionality or services provided by a system to users.

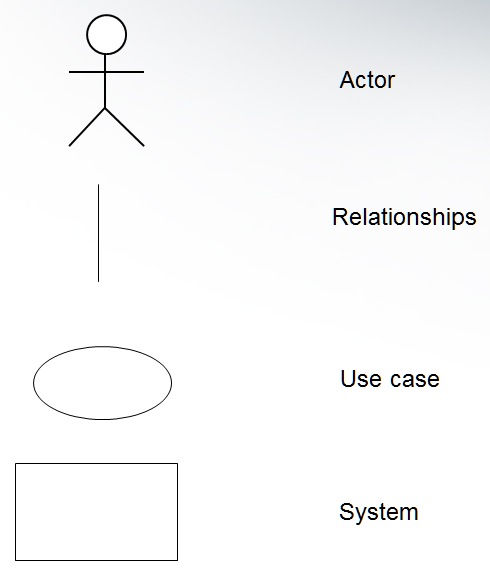


Fig 3.1.1

3.2 Class Diagram

Class Diagram depicts the relationship between classes, their attributes and methods. A class icon is simply a rectangle divided into three compartments. The topmost compartment contains the name of the class. The middle compartment contains a list of attributes (member variables), and the bottom compartment contains a list of operations (member functions). In many diagrams, the bottom two compartments are omitted.

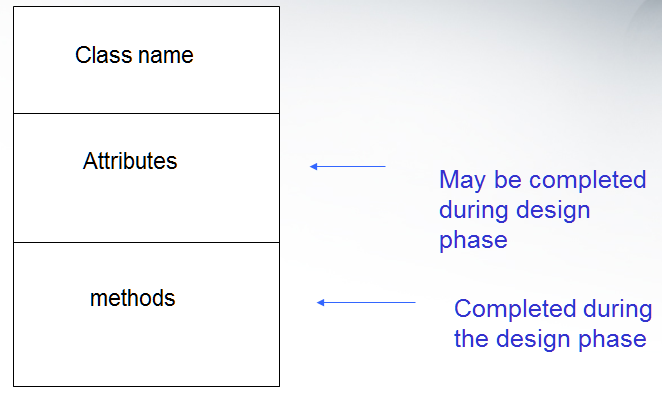


Fig 3.2.1

3.3 Sequence Diagram

Sequence Dig. describe the dynamic behavior between actors and the system and between objects of the system. Classes are represented by columns. Messages are represented by arrows, Activations represented by narrow rectangles and Lifelines are represented by dashed lines

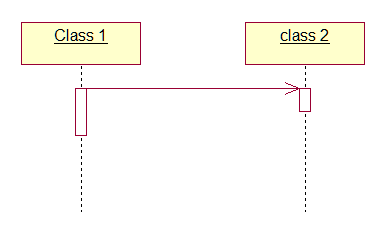


Fig 3.3.1

3.4 Activity Diagram

Activity Diagrams are used to represent the behavior of a system in terms of activities and their precedence constraints. These are compared to flowchart diagrams because, they can be used to represent control flow (order in which operations occur) and They can be used to represent data flow (objects exchanged among operations)

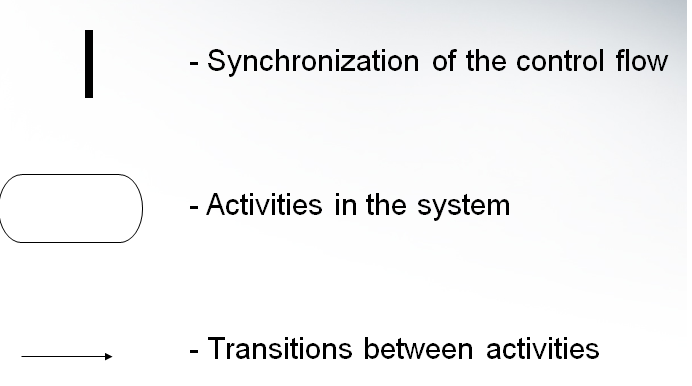


Fig 3.4.1

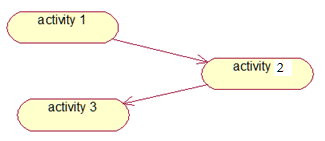


Fig 3.4.2

3.5 Component Diagram

Component diagrams describe the organization of and   
dependencies among software implementation components. These diagrams contain components, which represent distributable physical units, including source code, object code, and executable code.

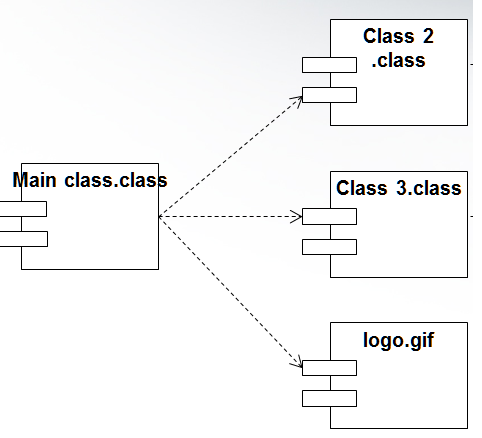


Fig 3.5.1

3.6 Flow Diagram

Flow Dig. of Component Based Software Eng. approach uses steps involved in the overall development.



Fig 3.6.1**4. Development of the Software:**

4.1 Design Phase

Using the theoretical tools discussed above, we are dividing design phase into components based upon them.

4.1.1 Designing of Use-Case Diagram

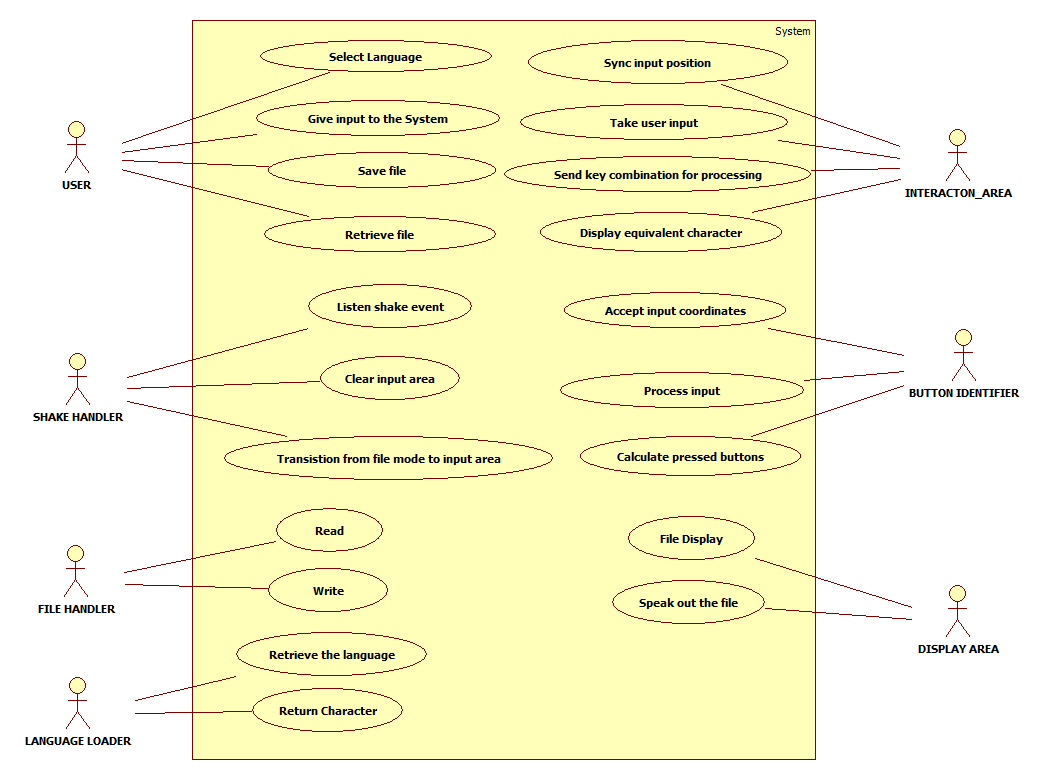


Fig 4.1.1.1

**Actors and their Use-Cases-**

**User –**

First of all when user starts the application, he can choose the language. Now after choosing the language, the input to the system is given through touch screen. After the desired text has been written, the file can be saved with a file name. The user can also access previously saved files through touch gestures.

**Interaction Area –**

It includes basic application features like synchronization of the input position when user touches the screen, accept input from the user and display the corresponding character typed.

**Button Identifier –**

Based upon the original button positions, this decides the current pressed buttons.

**Shake Handler-**

It does the work of switching of screen from file mode to write mode and to clear the input area.

**File Handler-**

File Handler handles file IO requests.

**Display Area-**

It simply displays the contents of the file it just read.

**Language Loader-**

It handles the language change requests from the user.

4.1.2 Designing of Class Diagram

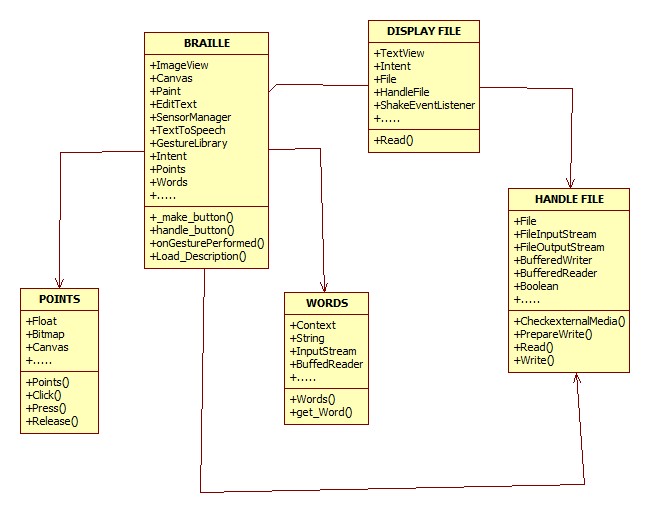


Fig 4.1.2.1

**Braille-**

This class is in direct interface with all of the user interactions. It contains objects of all of the other classes (to call them, when needed). Braille class also contains interaction objects and drawing objects like ImageView, Canvas, GestureLibrary etc. It contains objects of Points, Words, DisplayFile and Handle File class.

**Words-**

This class handles the language loading and determination of character based upon the user input combination. It return the character to the Braille class(which is the calling class) through method get\_word().

**Points-**

Points handles the determination of pressed button combination based upon the original center position. It also informs the user about the button pressed by a color change in pressed button.

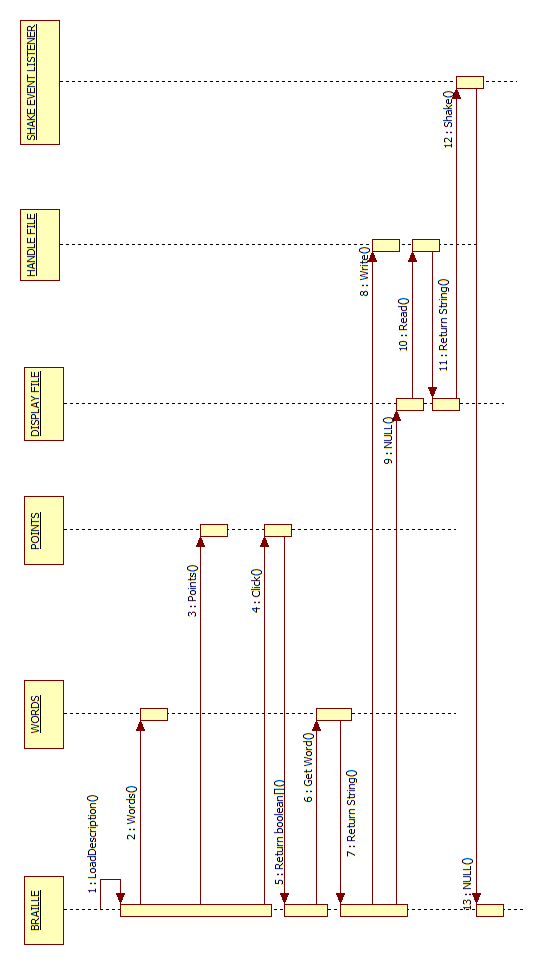
**DisplayFile-**

This class simply displays the string returned be HandleFile class to the TextView. It also uses ShakeEventListener for going back to the parent calling class(Braille).

**HandleFile-**

HandleFile contains objects of File, FileInutStream/OutputStream. It sequentially reads the files saved in the file system and return the string to the calling class which may be Braille or DisplayFile.

4.1.3 Designing of Sequence Diagram

  
 Fig 4.1.3.1

**1:** LoadDescription() method tells the system about number of languages and their corresponding file names. All these details are saved into variable.

**2:** Constructor of Words, Words() is called to load the languages into its variables (used for later calculations) using the data returned by LoadDescription. It loads the text files describing the languages. The characters and their corresponding key-mapping values are loaded into the system.

**3:** Constructor of Points, Points() is called to store the initial center co-ordinates of user pressed buttons. These would be used to determine location of button pressed later during the user interaction.

**4:** As soon as user press some combination, method Click() of Points is called, this method returns a valid combination of buttons using the previously stored center co-ordinates.

**5:** This is the Boolean array returned by the click() method.

**6:** Using the Boolean array, get\_word() method of Words is called which calculate the corresponding character from its previously saved variables.

**7:** This is the string returned by the get\_word() method. It contains a character and its pronunciation.

**8:** When user wants to save the file, Write() method is called and the typed string is passed as an parameter to it.

**9:** When user enters into file mode, DisplayFile activity is instantiated and control shifts to this activity now.

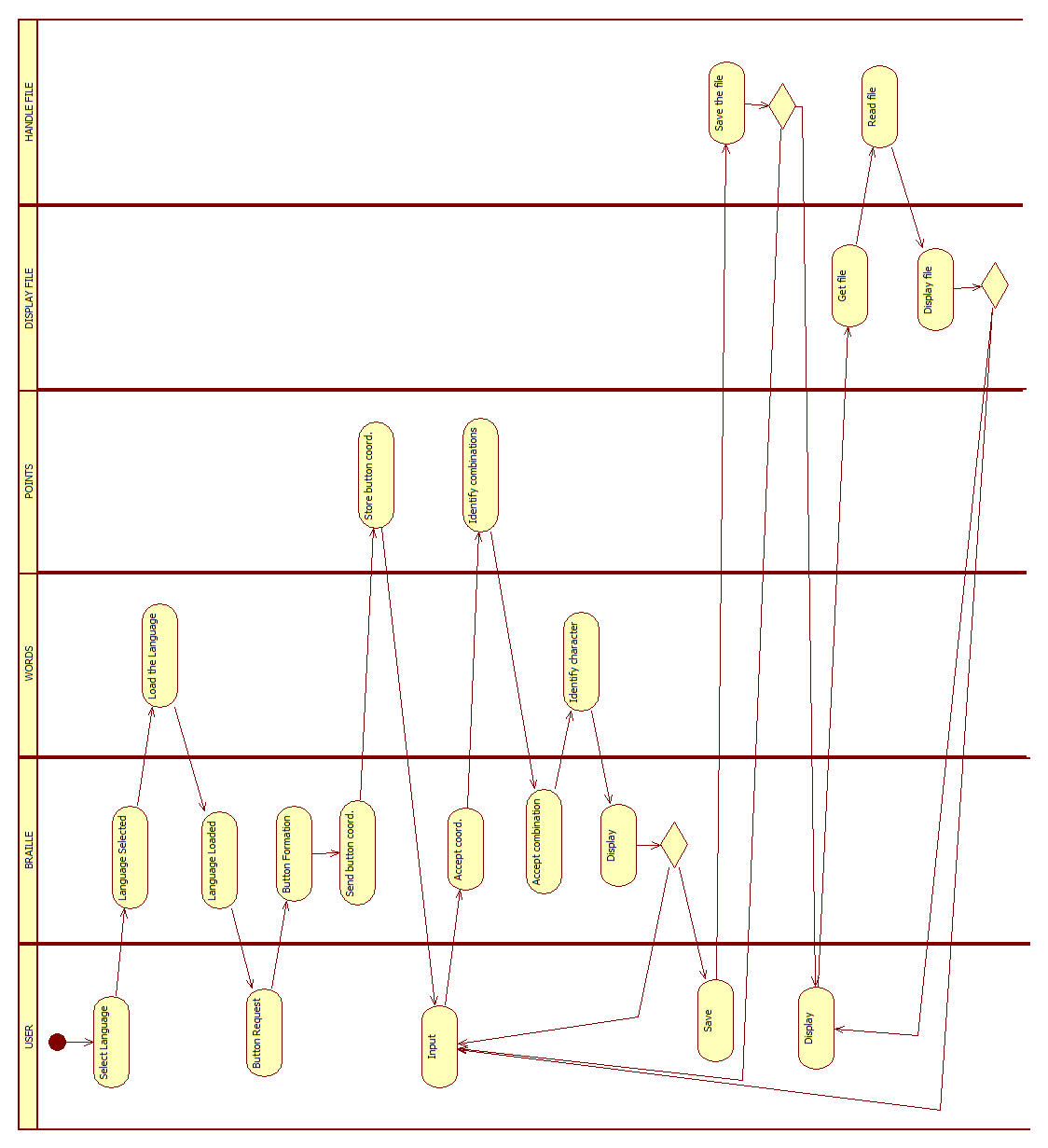
**10:** Read() method of HandleFile is called to read the data from the file system of the Android. It return the data of next/previous file.

**11:** This is the string returned by the Read() method.

**12:** When user wants to go back to input area for entering more data, device is shaked and ShakeListener() method is called.

**13:** Shake() method takes the control back to the Braille class and now everything starts from beginning.

4.1.4 Designing of Activity Diagram

  
 Fig 4.1.4.1

The flow of activities goes as follows –

1. Language selection.
2. Language loading
3. Fresh button formation
4. Storing of button co-ordinates
5. Touch event/input
6. Identification of valid combination
7. Calculation of character based upon the combination
8. Display the character
9. Save/write
10. Display mode
11. Read the file
12. Display to TextView area
13. Repeat from step v or xi.

The diagram shows the sequence in which various activities of the system are being called and interacted upon. Simple sequence of events is listed above in addition to explanation given for the previous diagrams.

4.1.5 Designing of Component Diagram

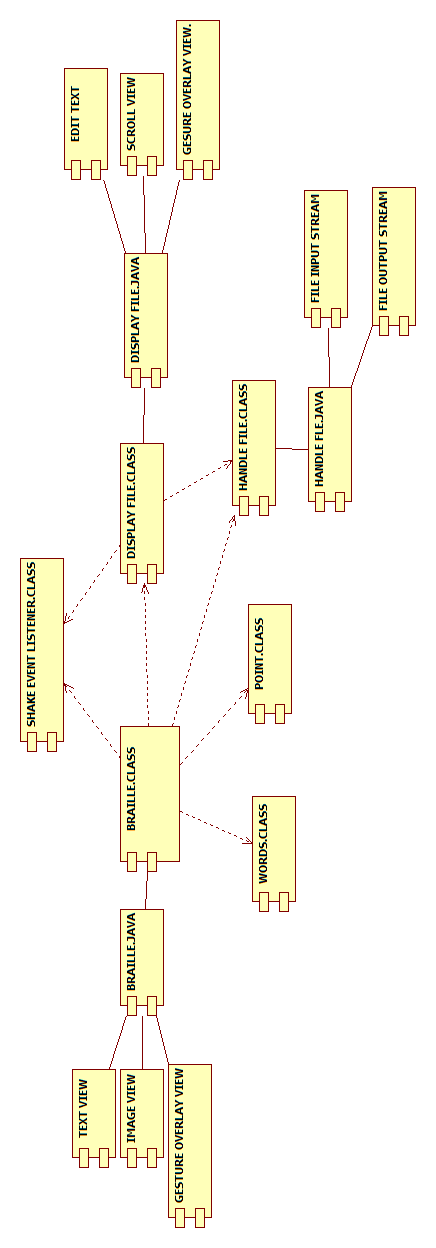


Fig 4.1.5.1

The major components of the software are depicted in the figure.

1. **Braille.java**

TextView, ImageView, GestureOverlayView, Braille.class …...

2. **Braille.class**

Words.class,Point.class,ShakeEventListener.class,DisplayFile.class, HandleFile.class …...

3.**DisplayFile.class**

HandleFile.class,DisplayFile.java, EditText, ScrollView, GestureOverlayView …...

4.**HandleFile.class**

HandleFile.java, FieOutputSteam, FileInputStream.

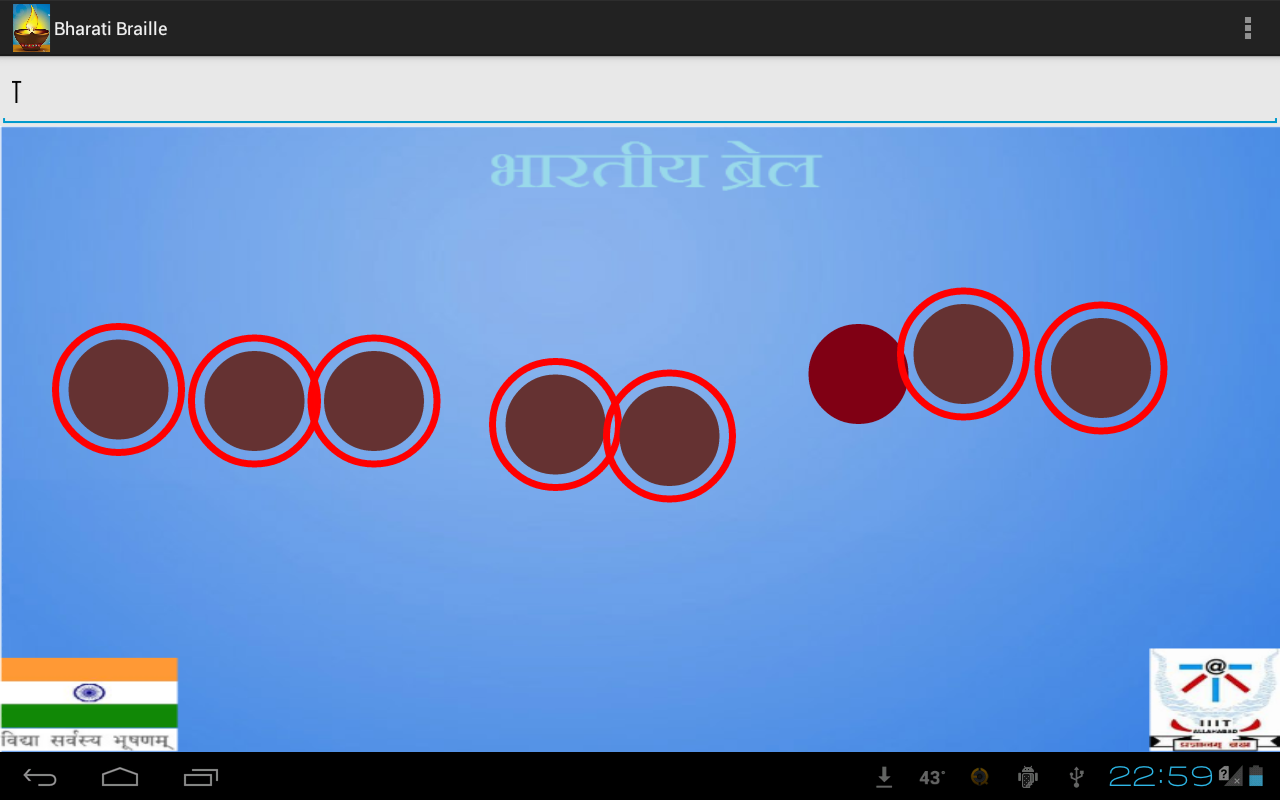
4.2 Implementation Phase

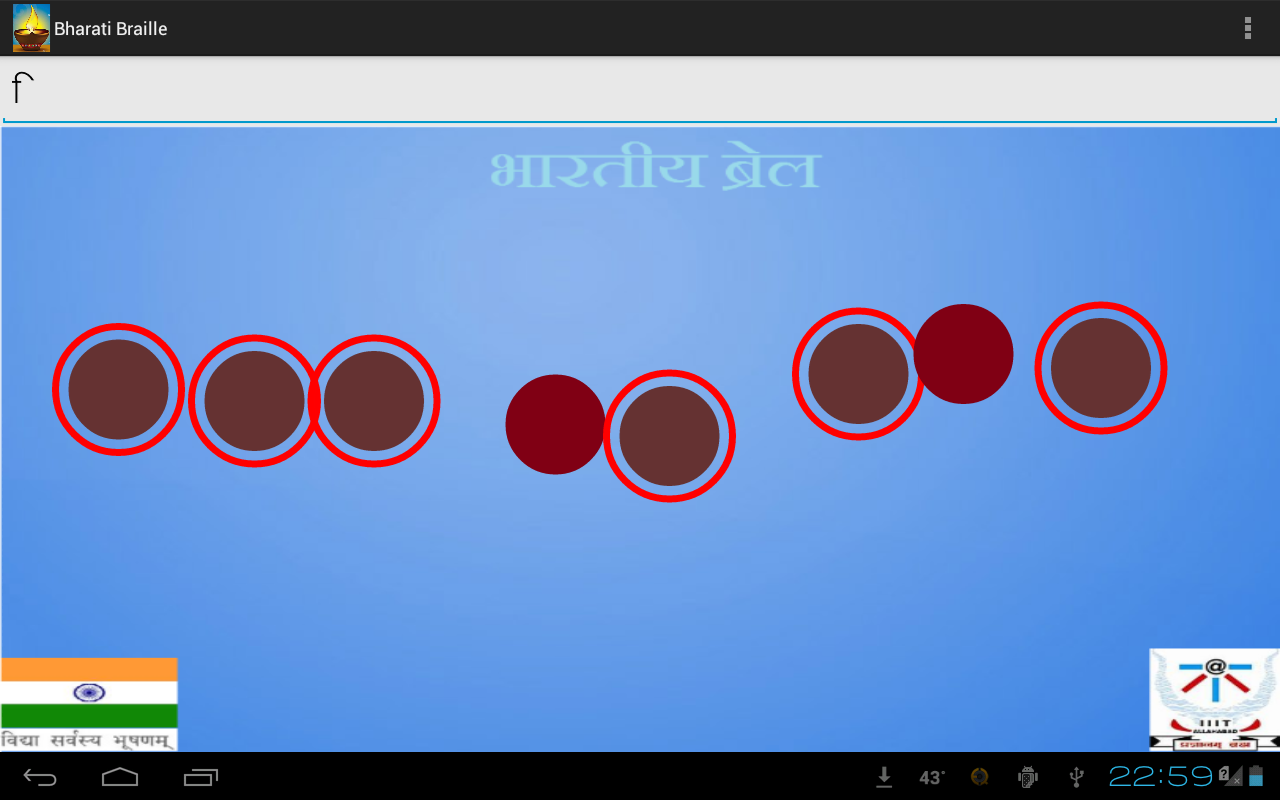
All the designs have been successfully implemented on Android.

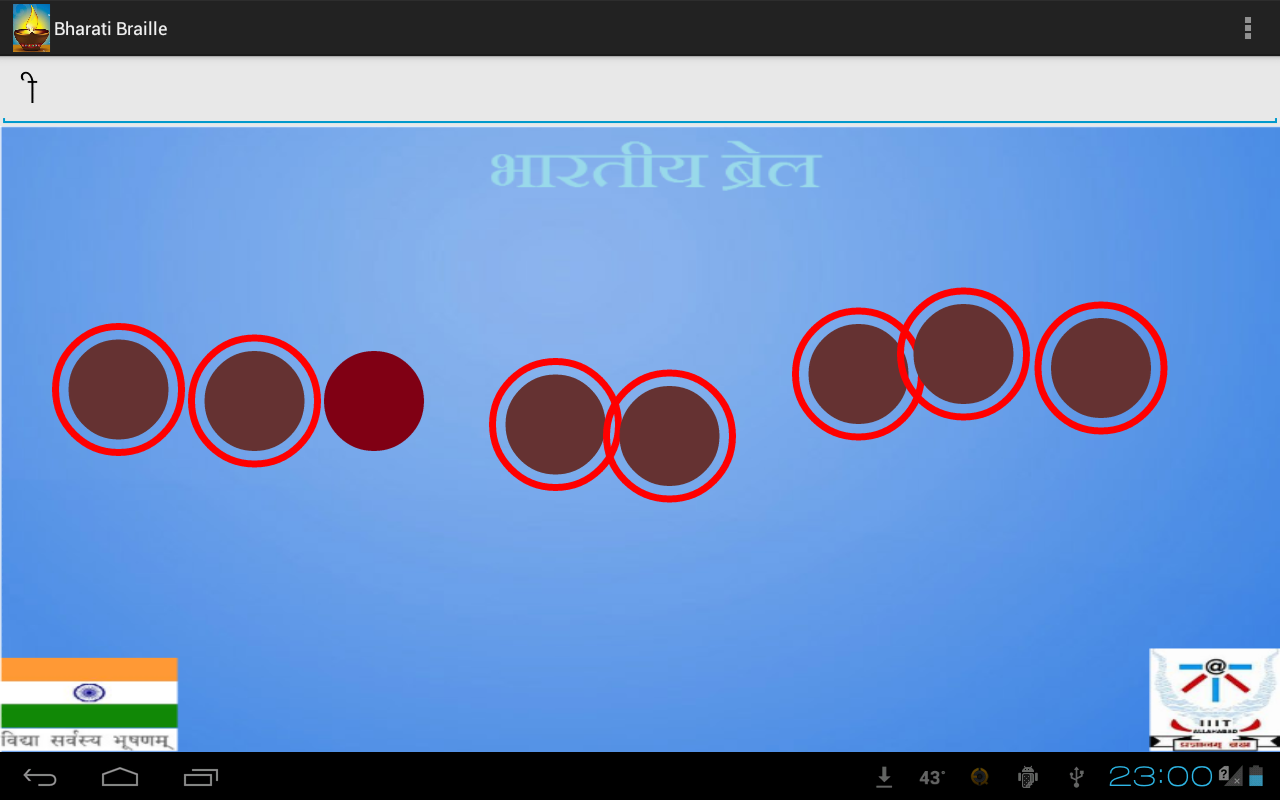
**5. Testing And Analysis**

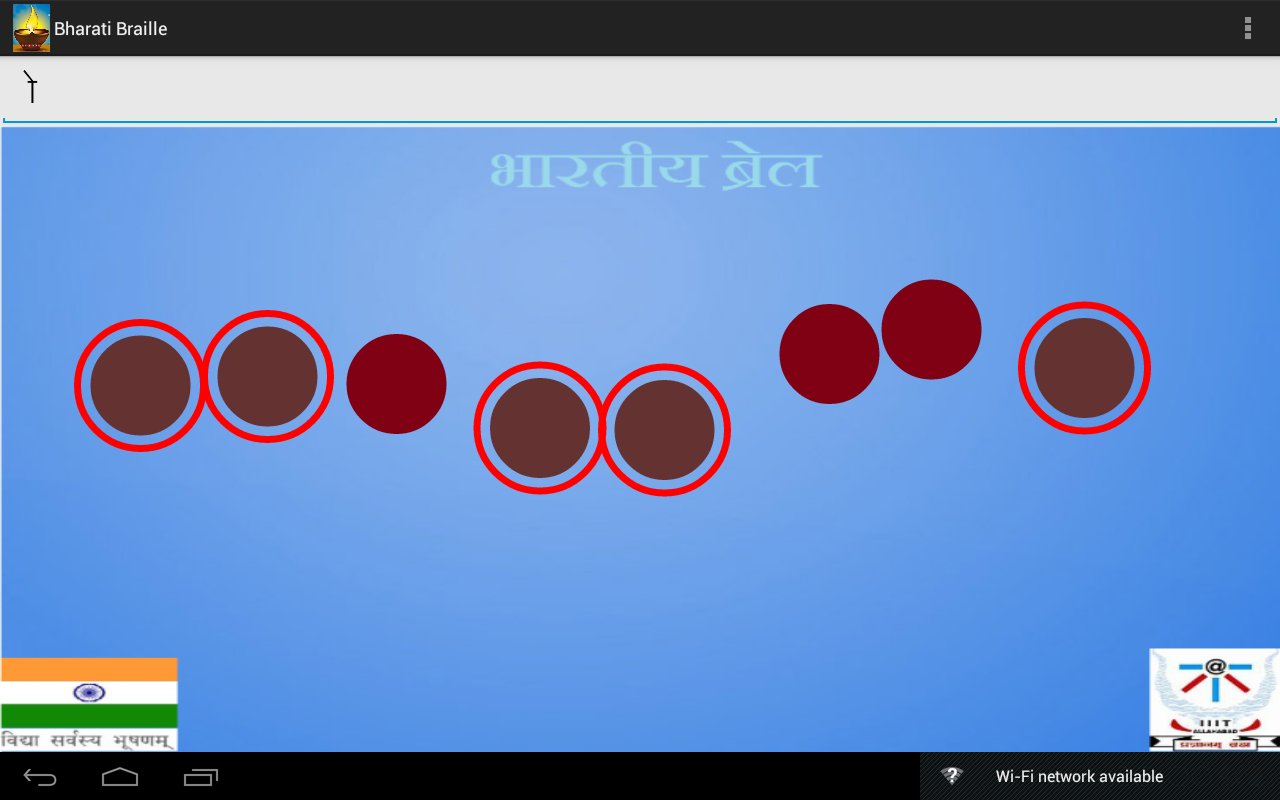
5.1 Testing

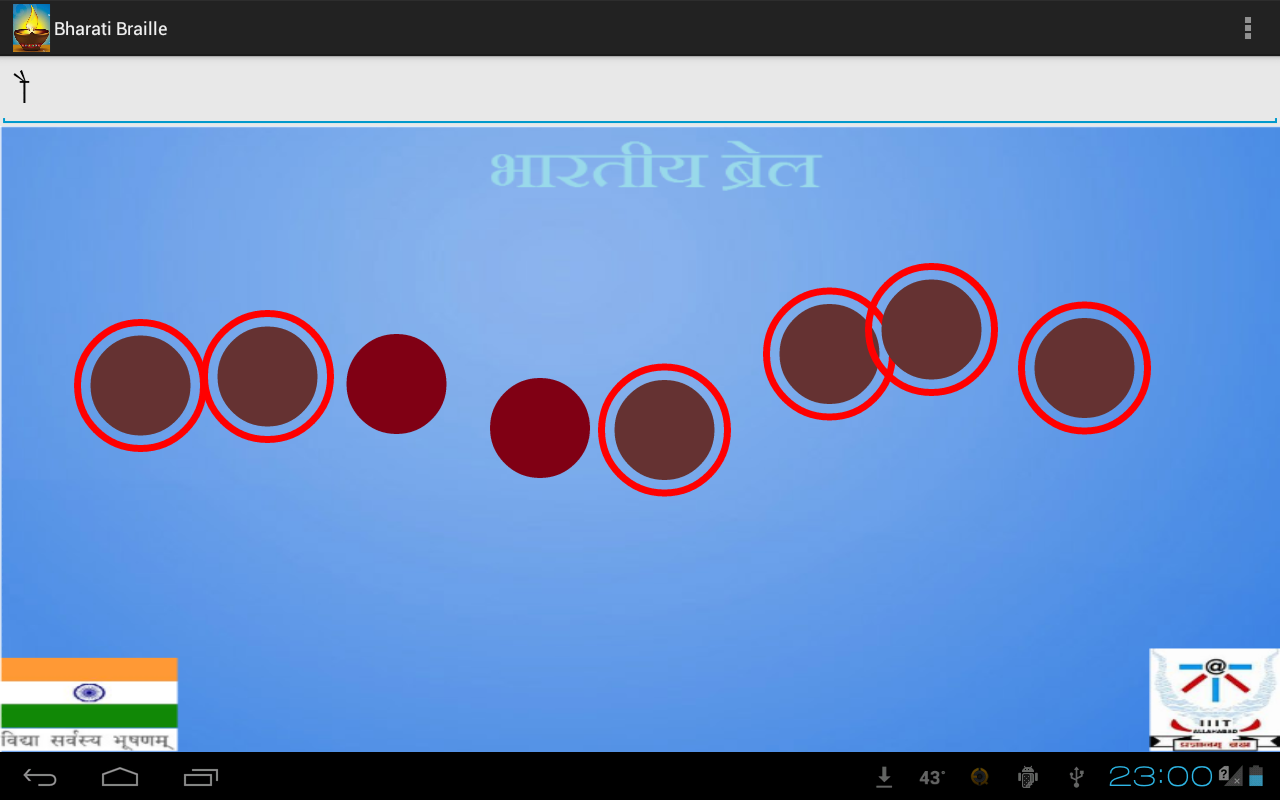
We aim to provide a generic Hindi Braille Simulatort herefore we have integrated “MATRAs” from our side and following are the button combinations for the same.

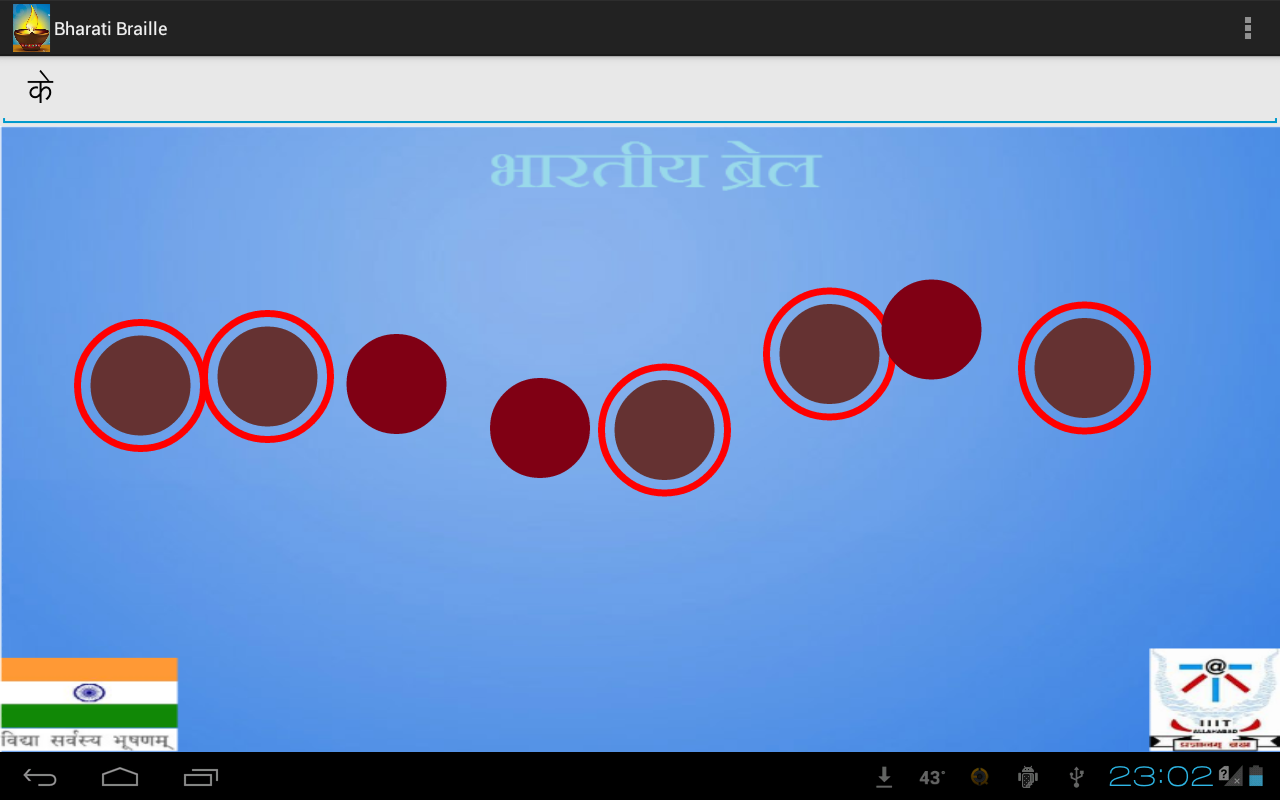


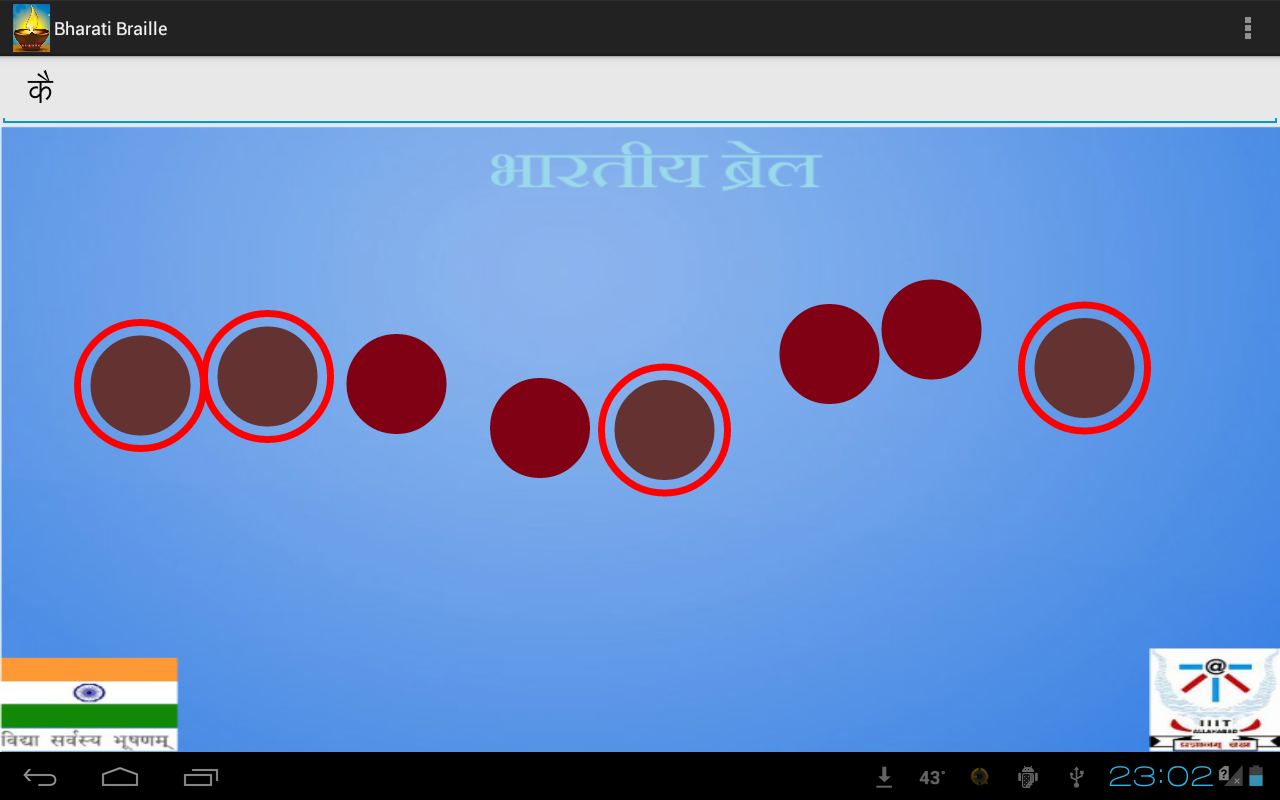


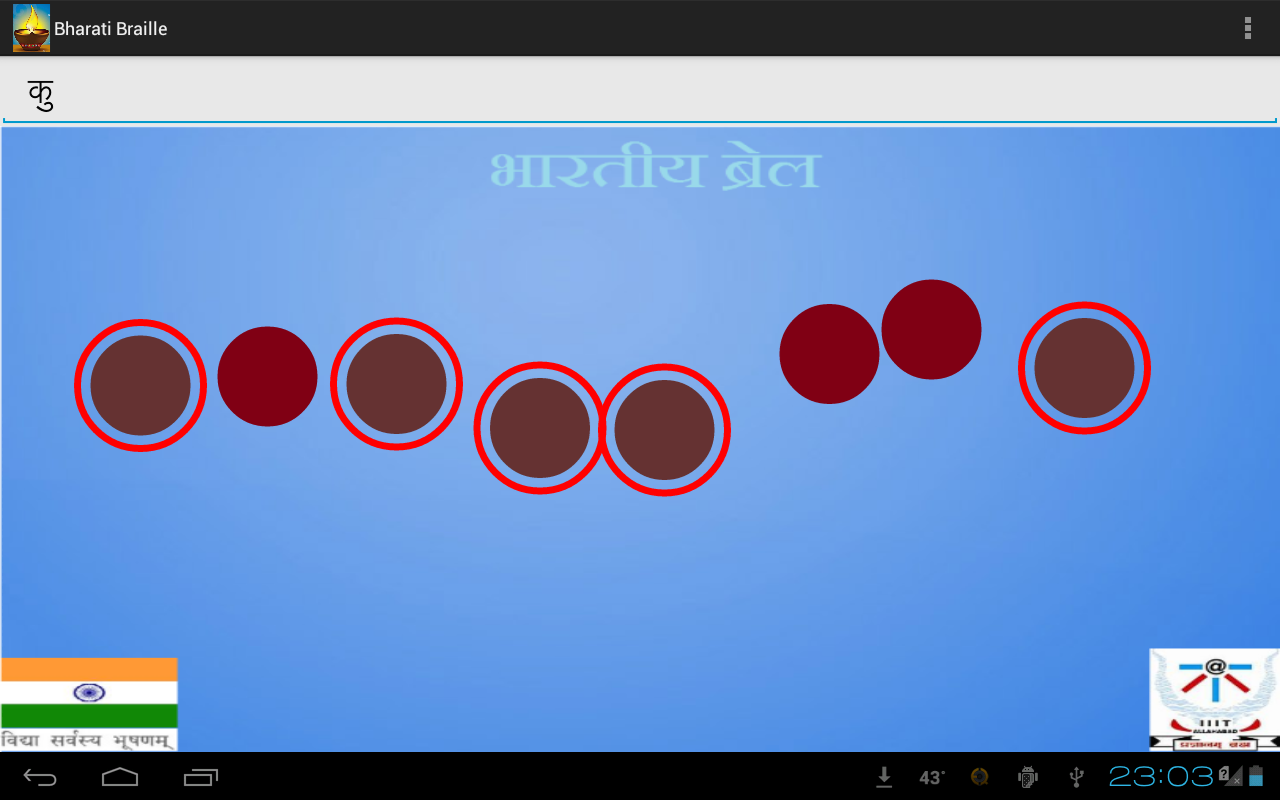


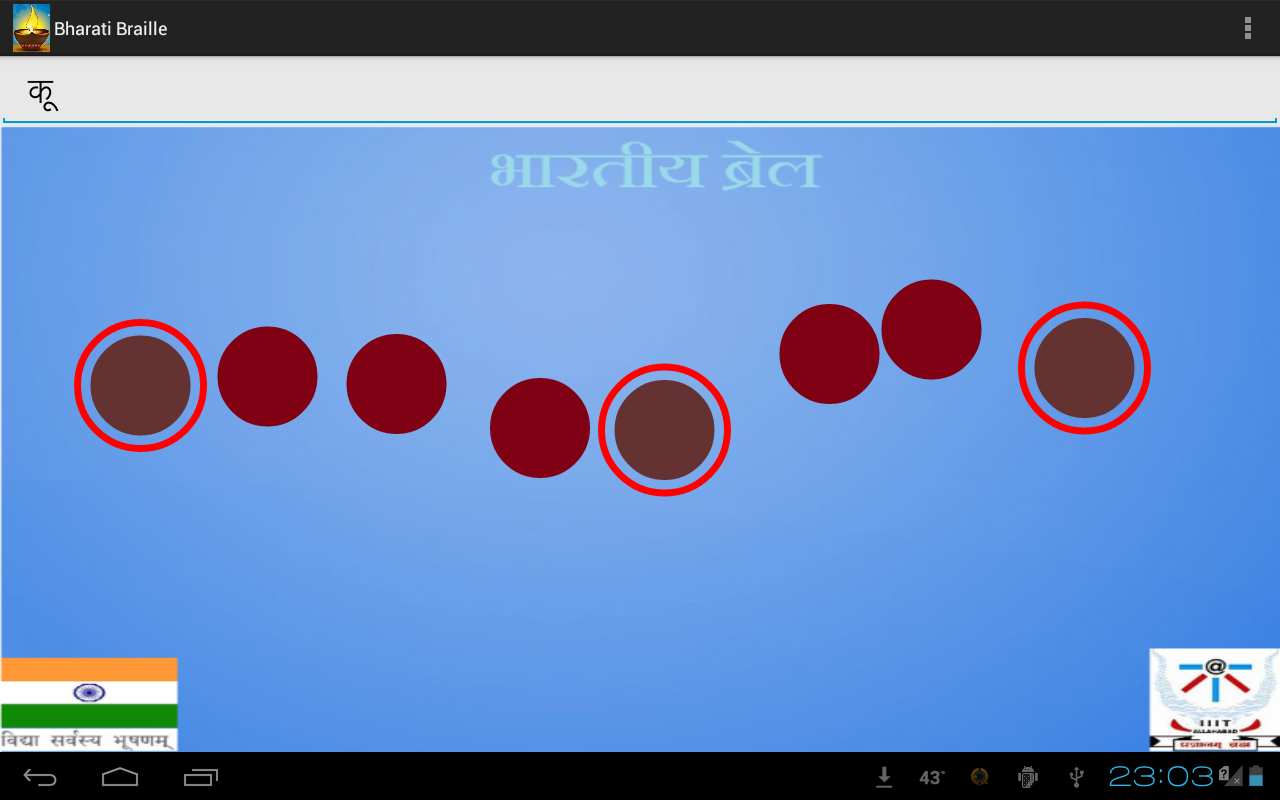




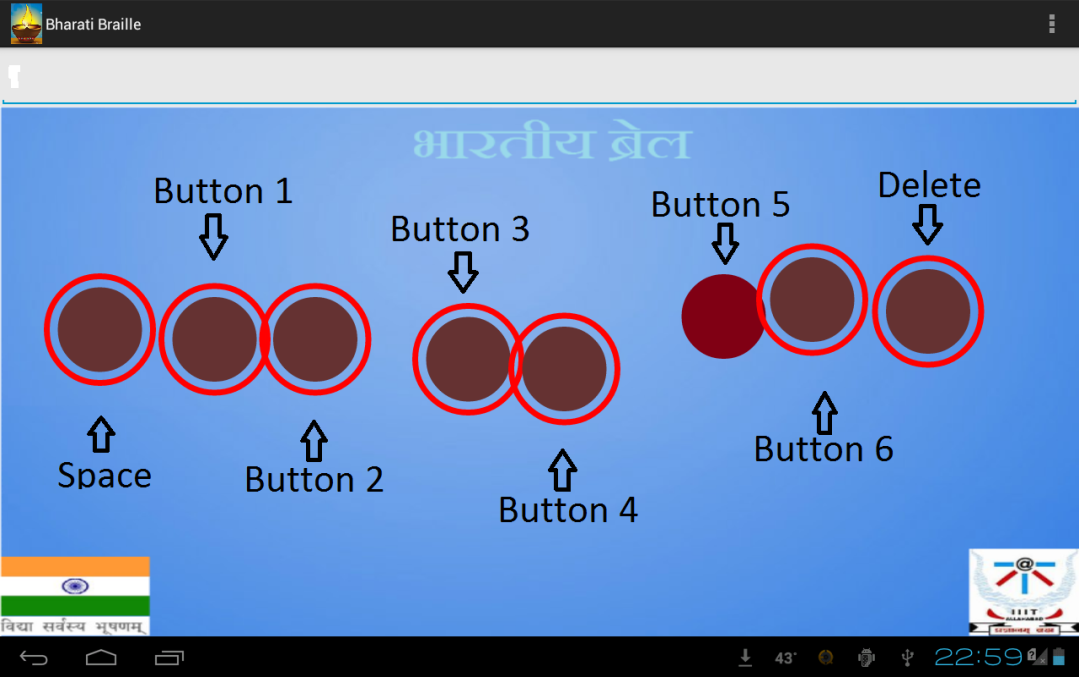


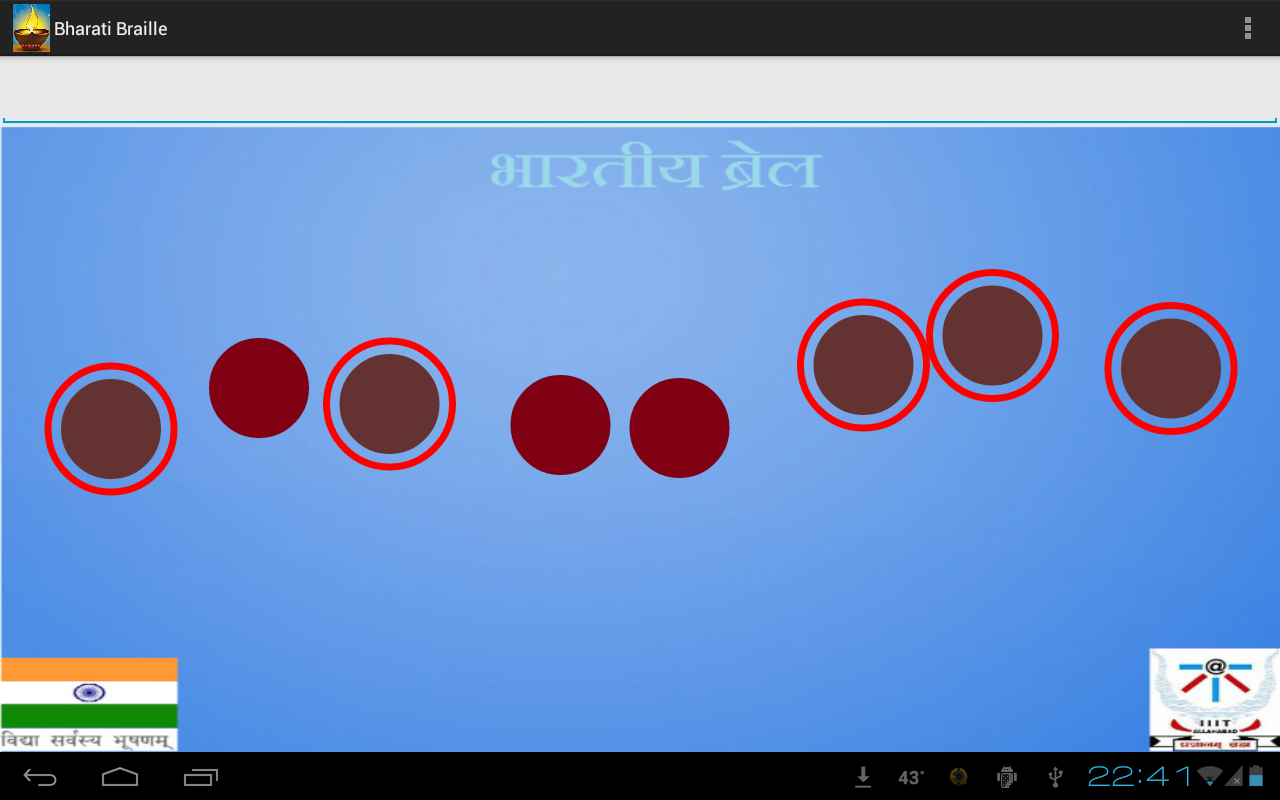


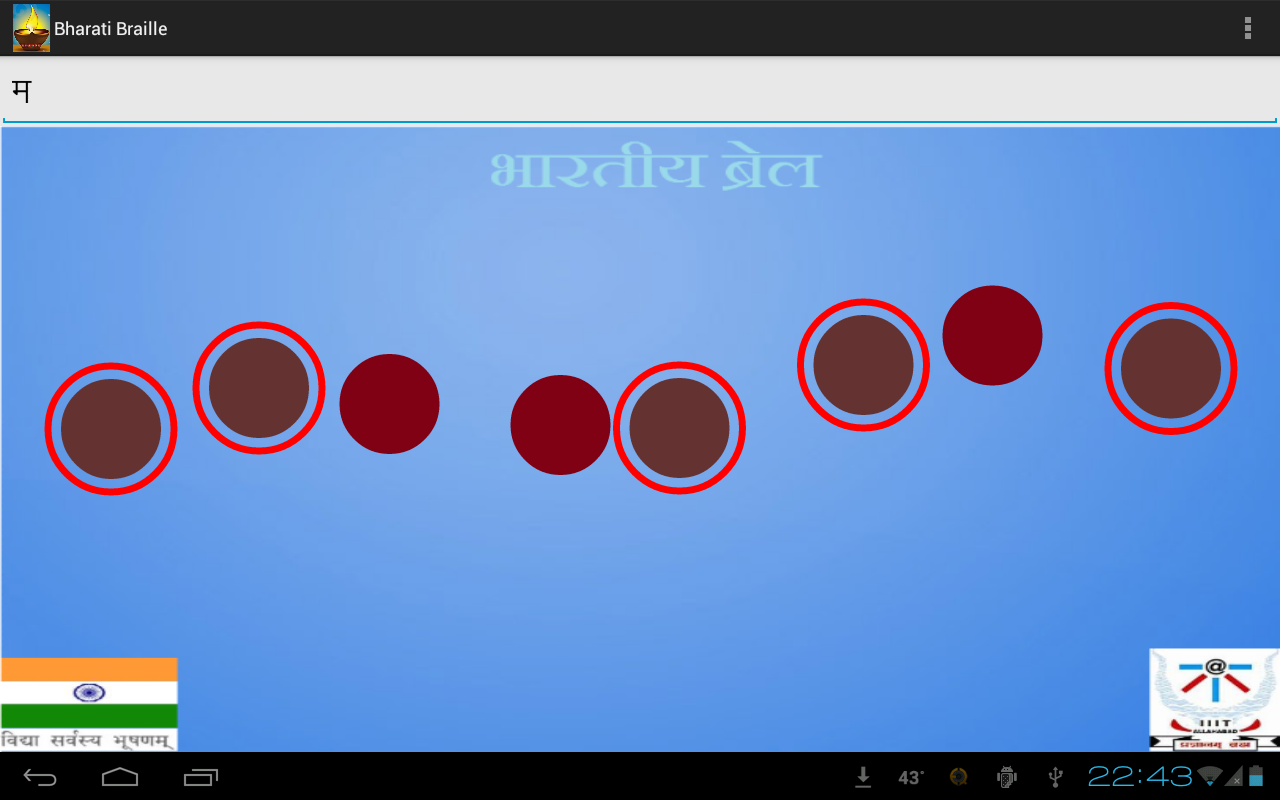


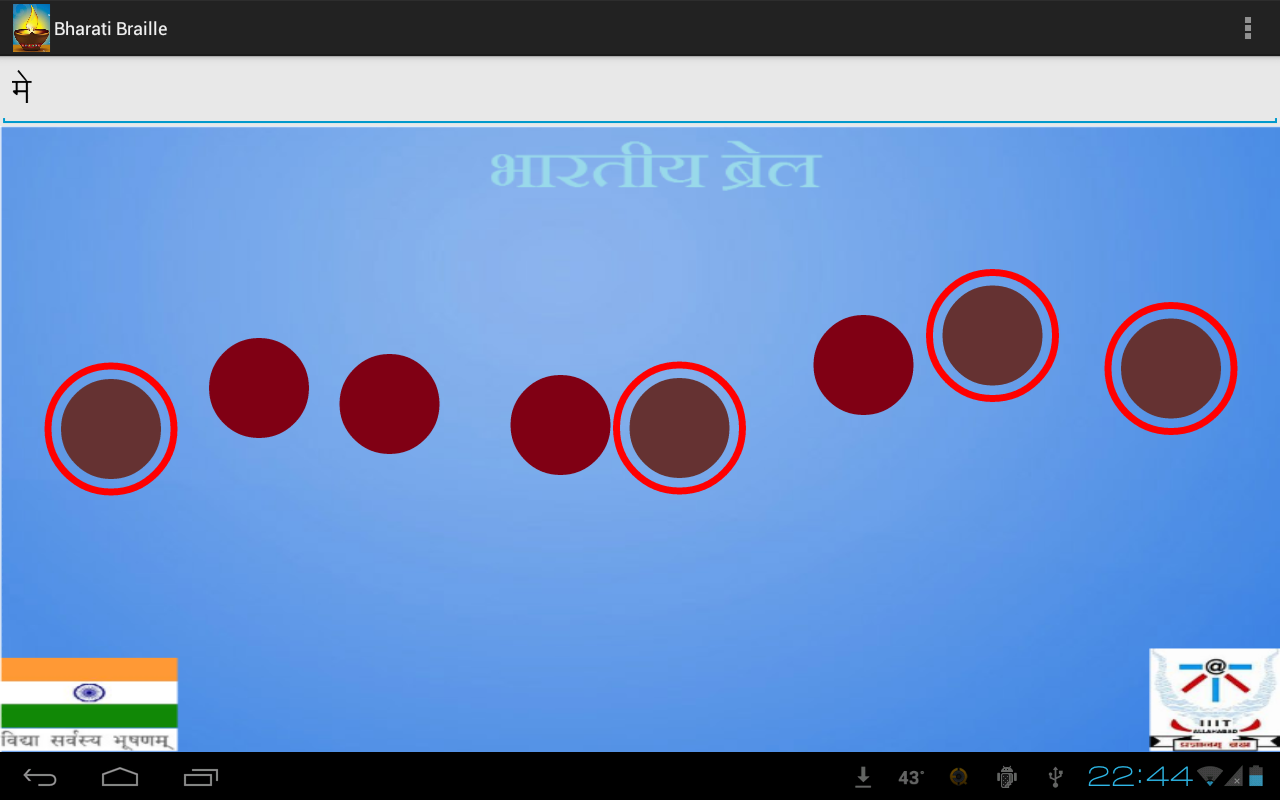


We have tested the application on a large set of words. Below snapshots demonstrates how our Hindi Braille Simulator works to generate a simple Hindi sentence.

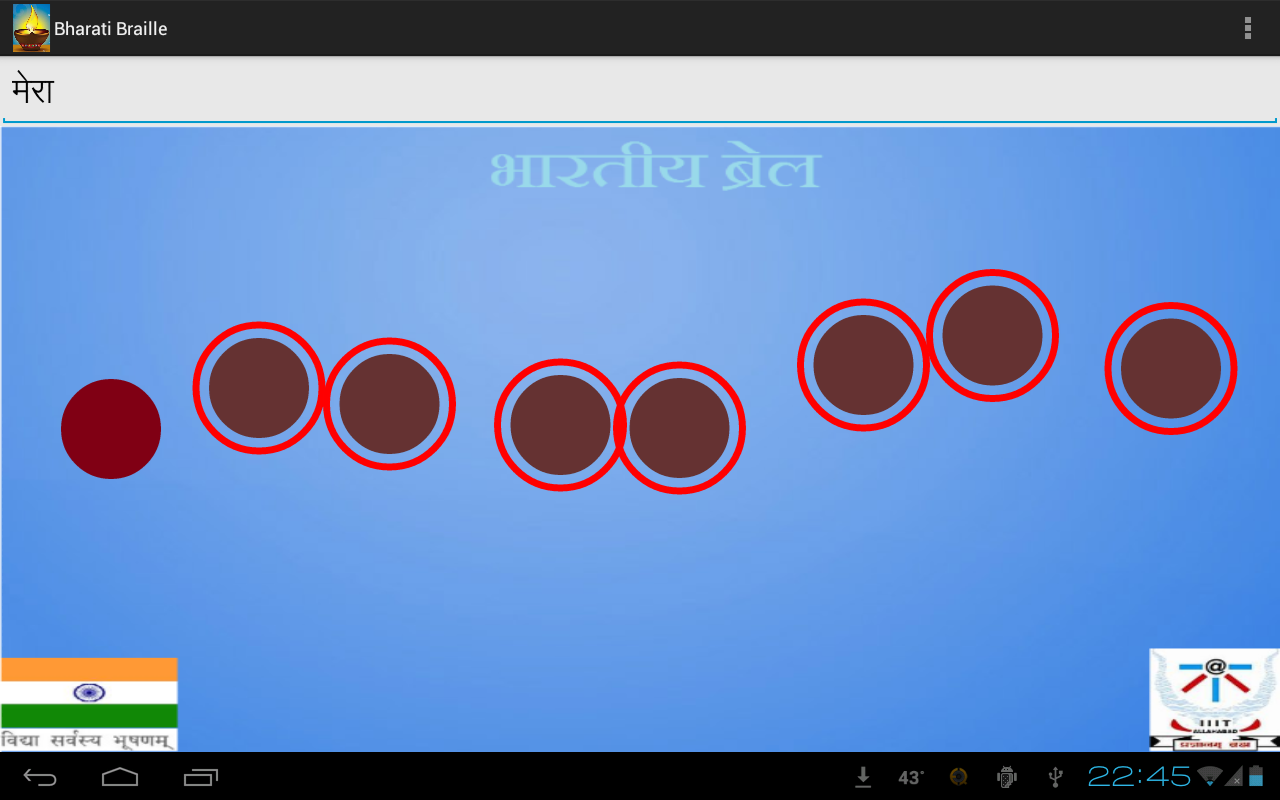


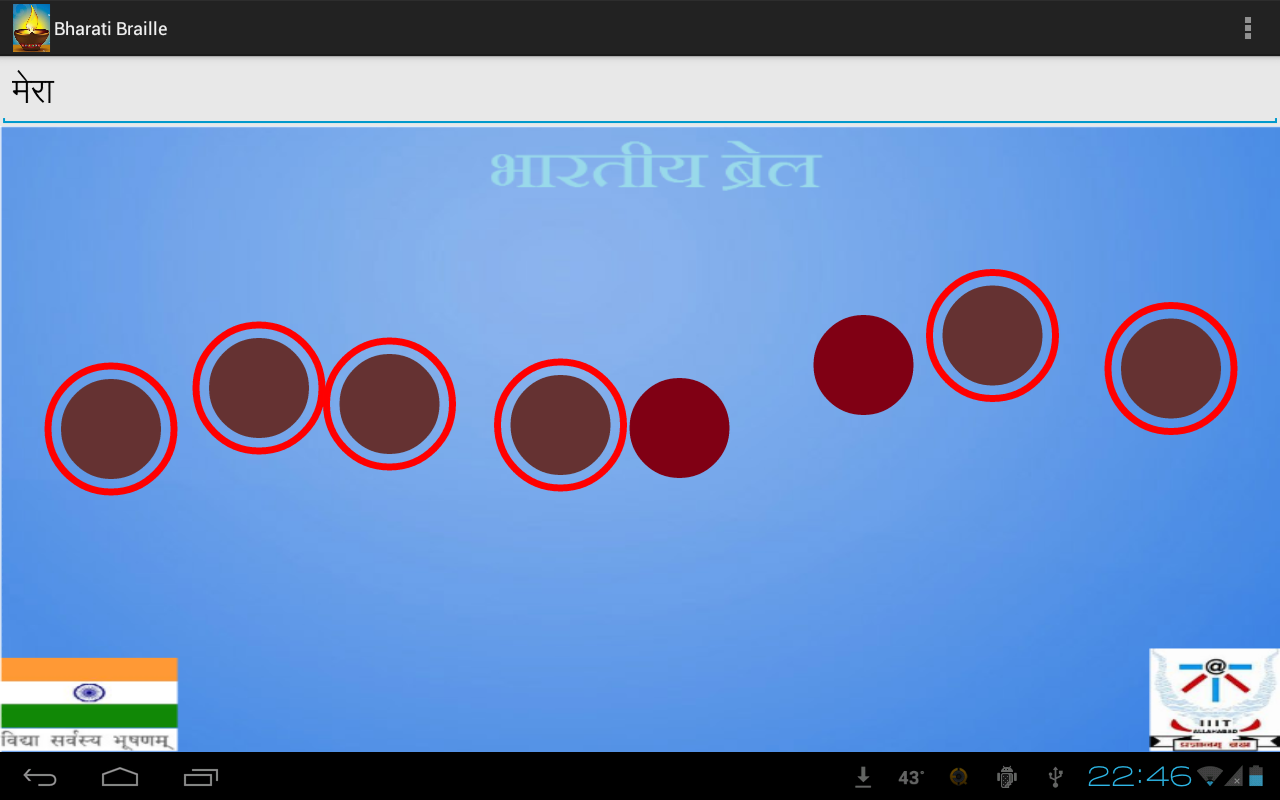


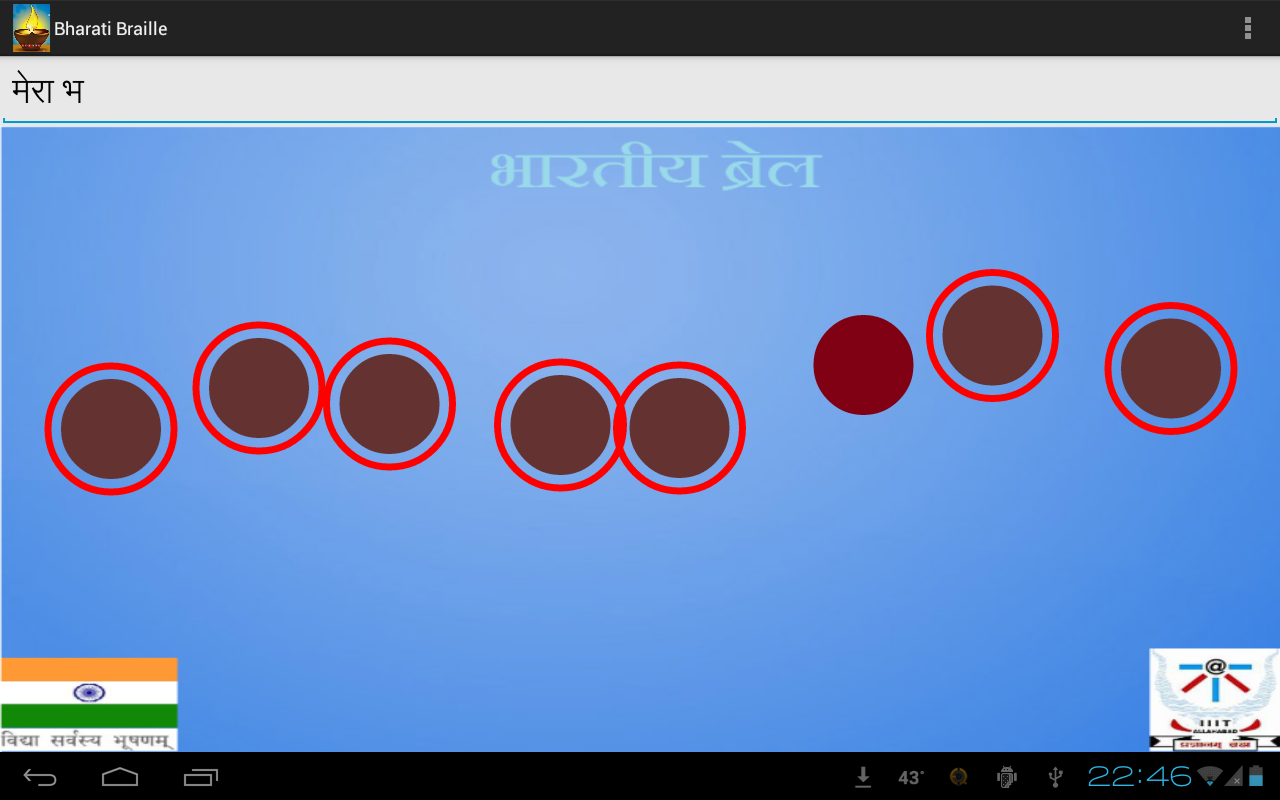


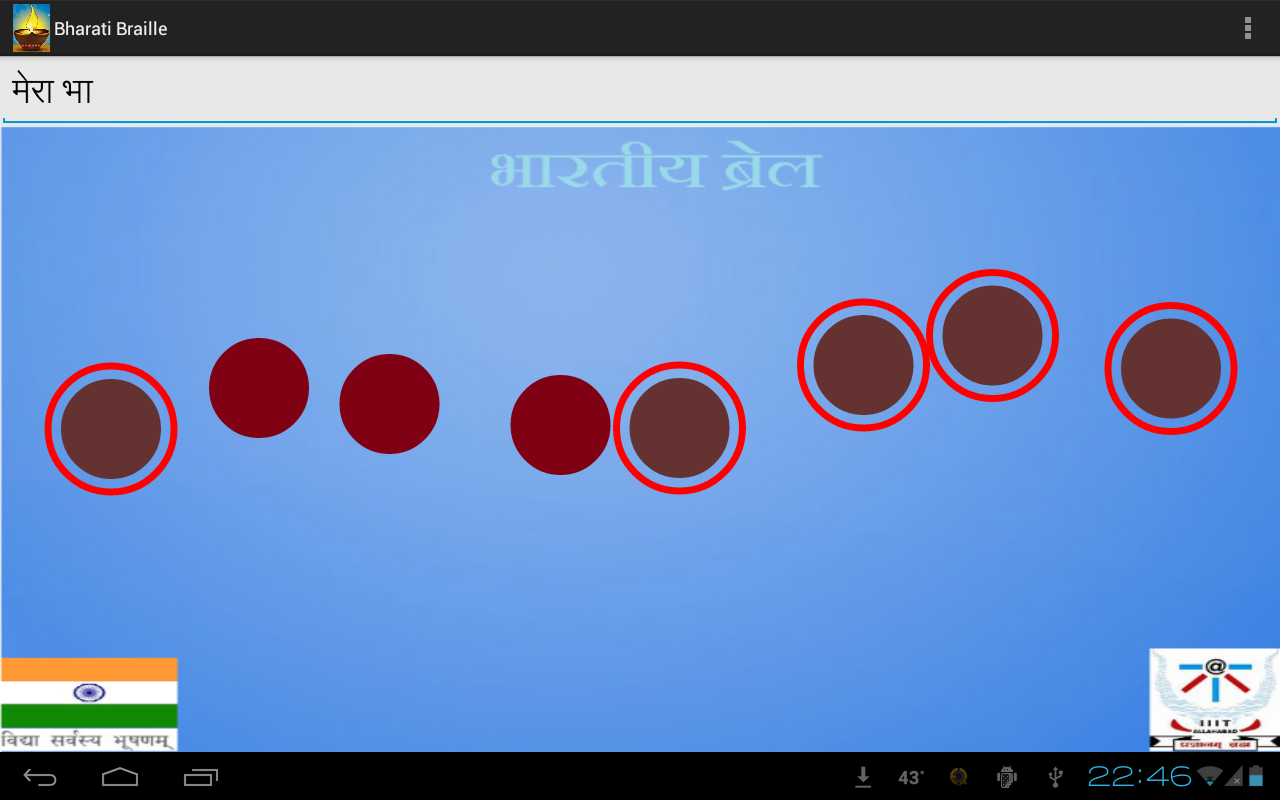


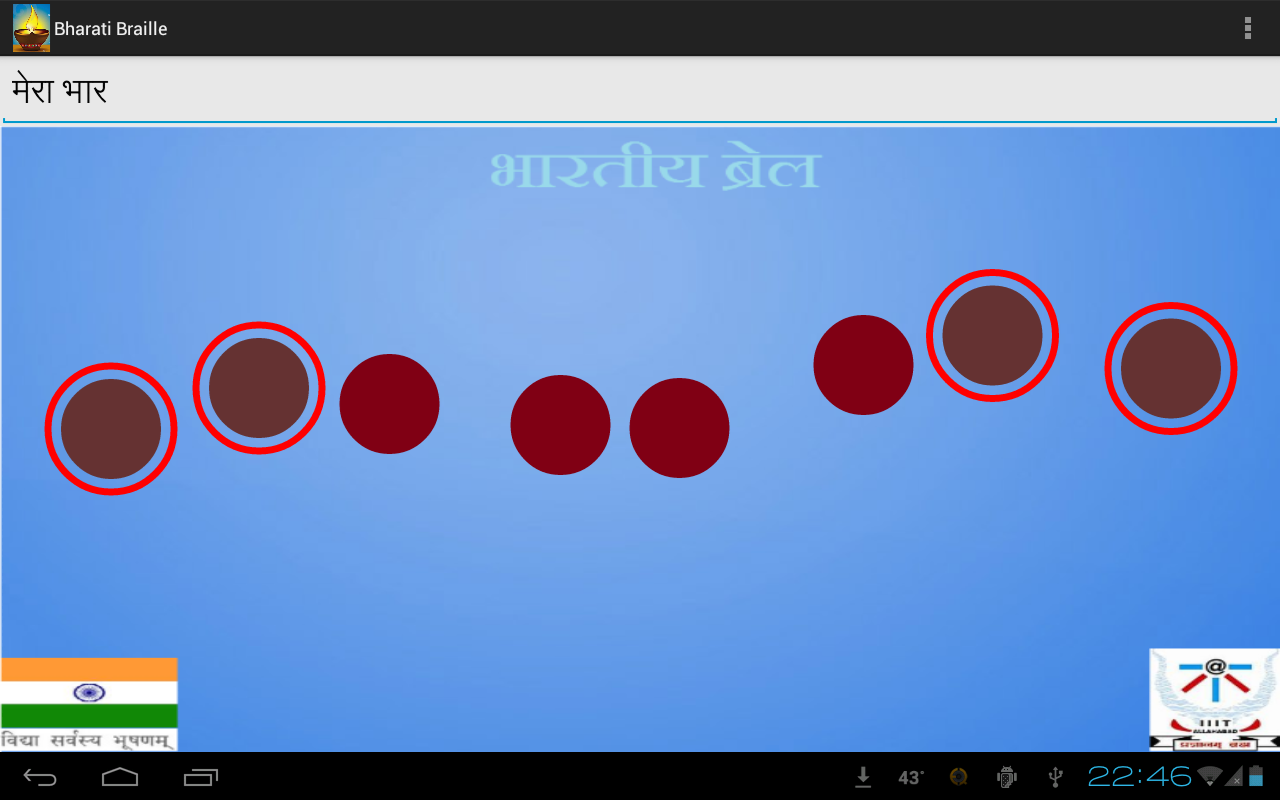


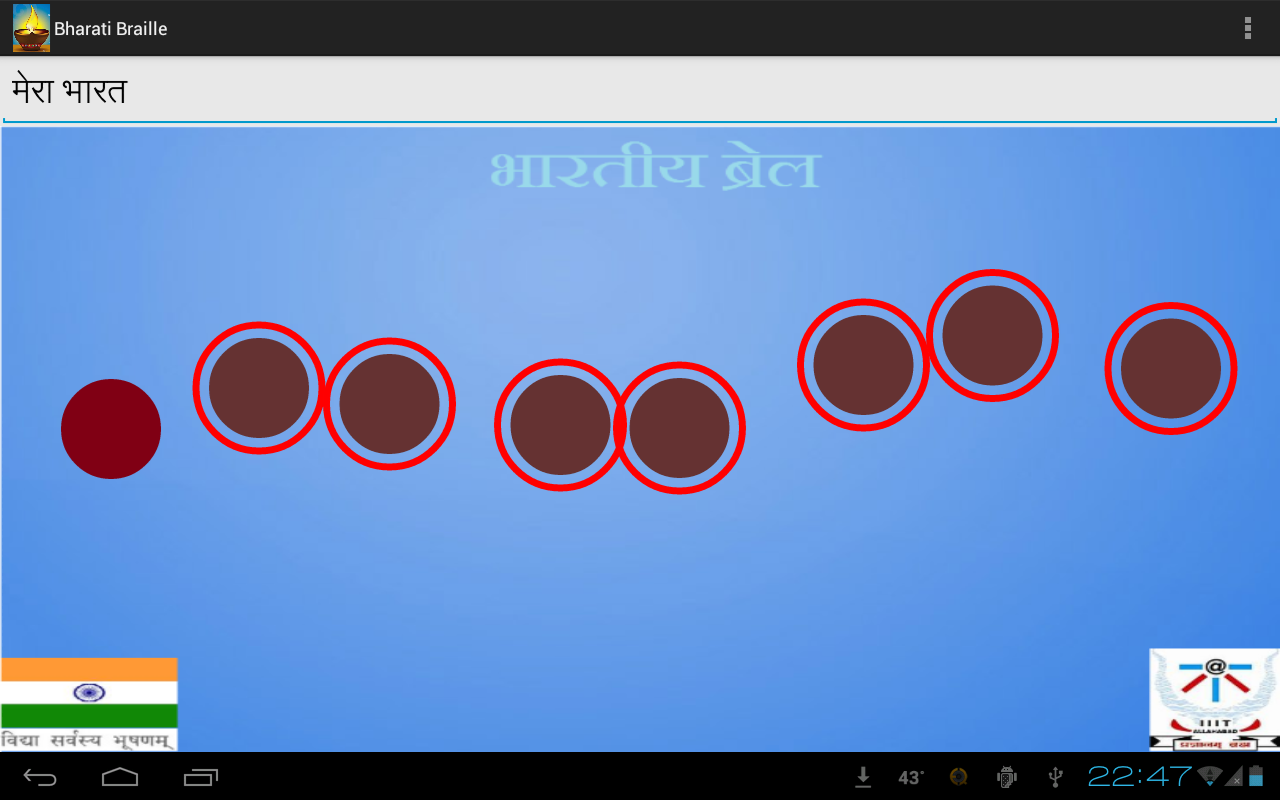


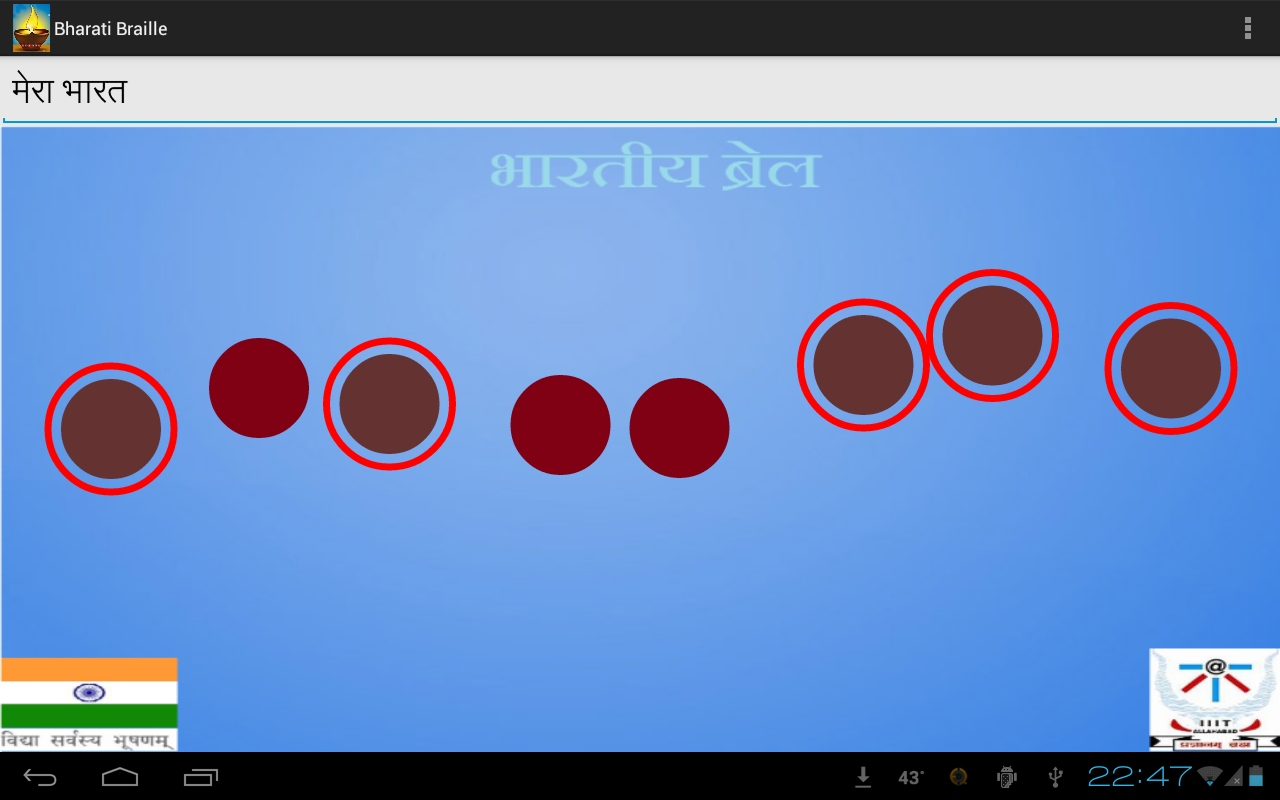


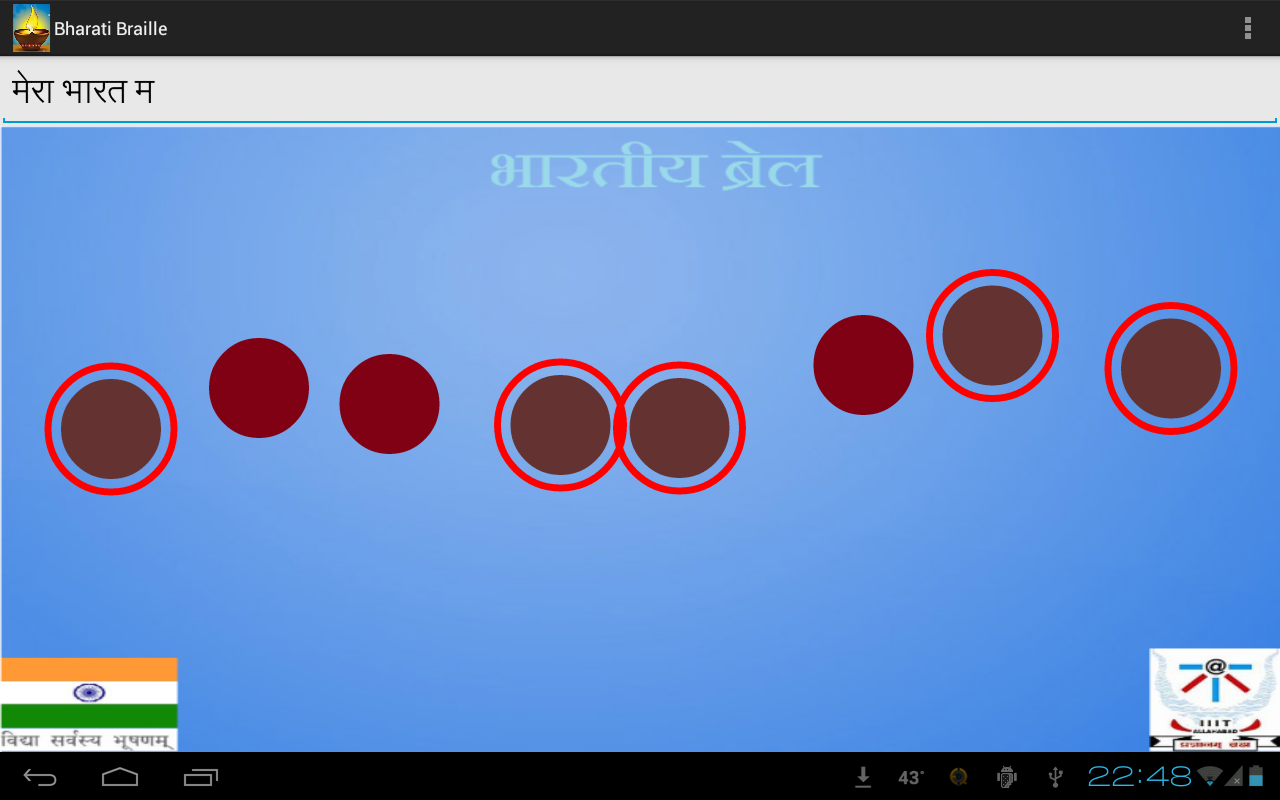


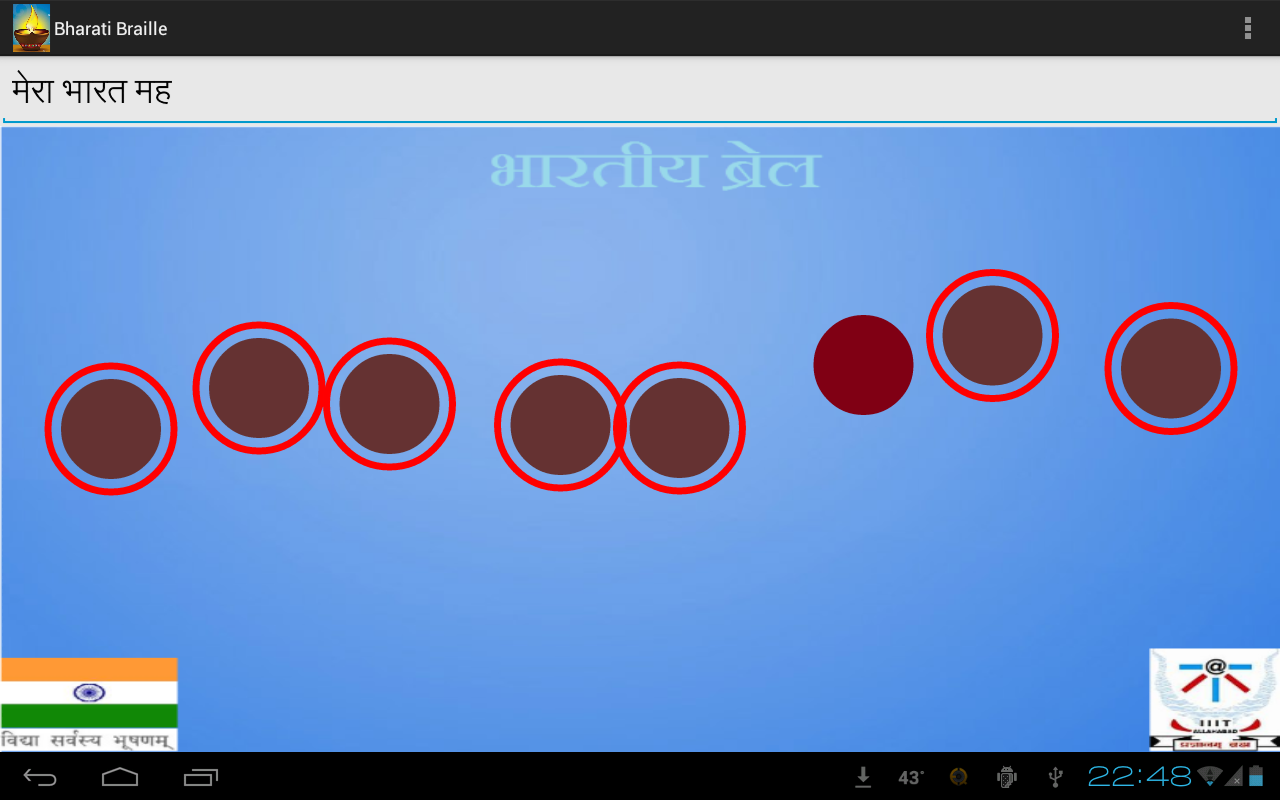


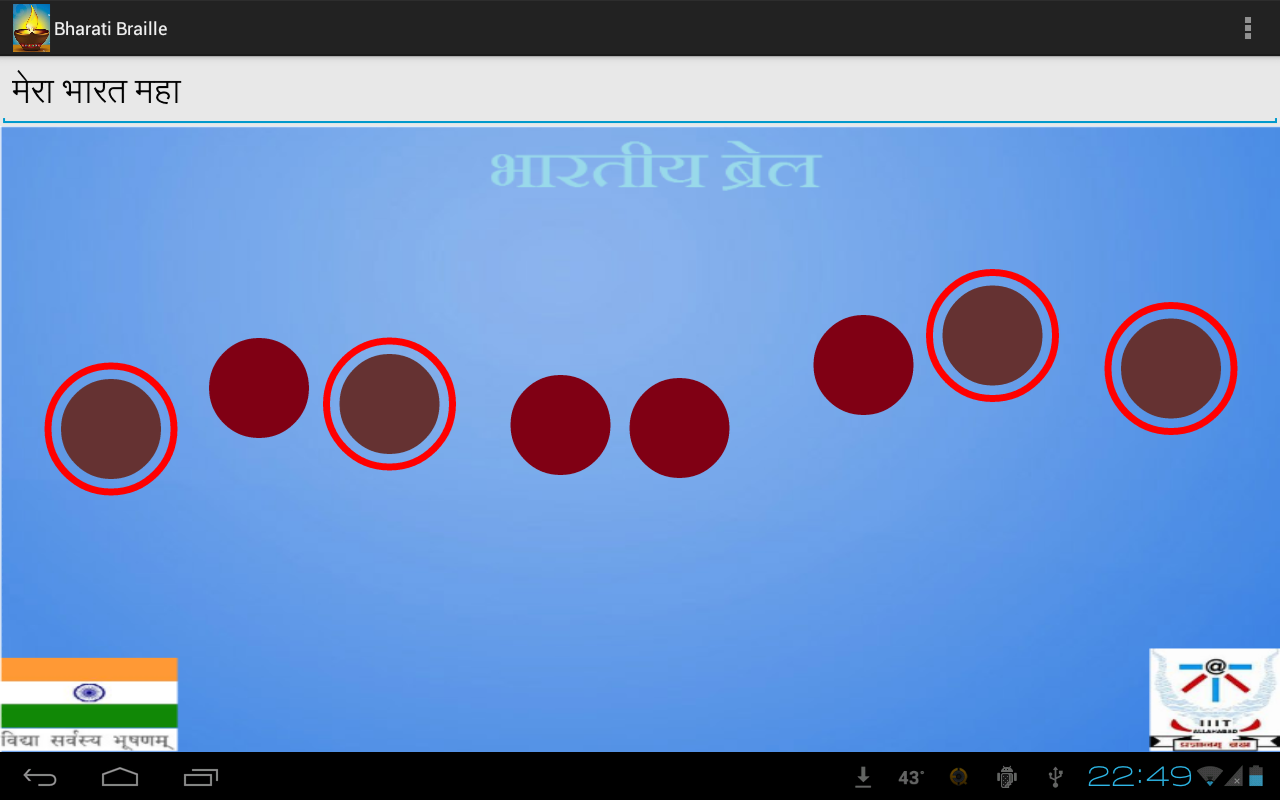


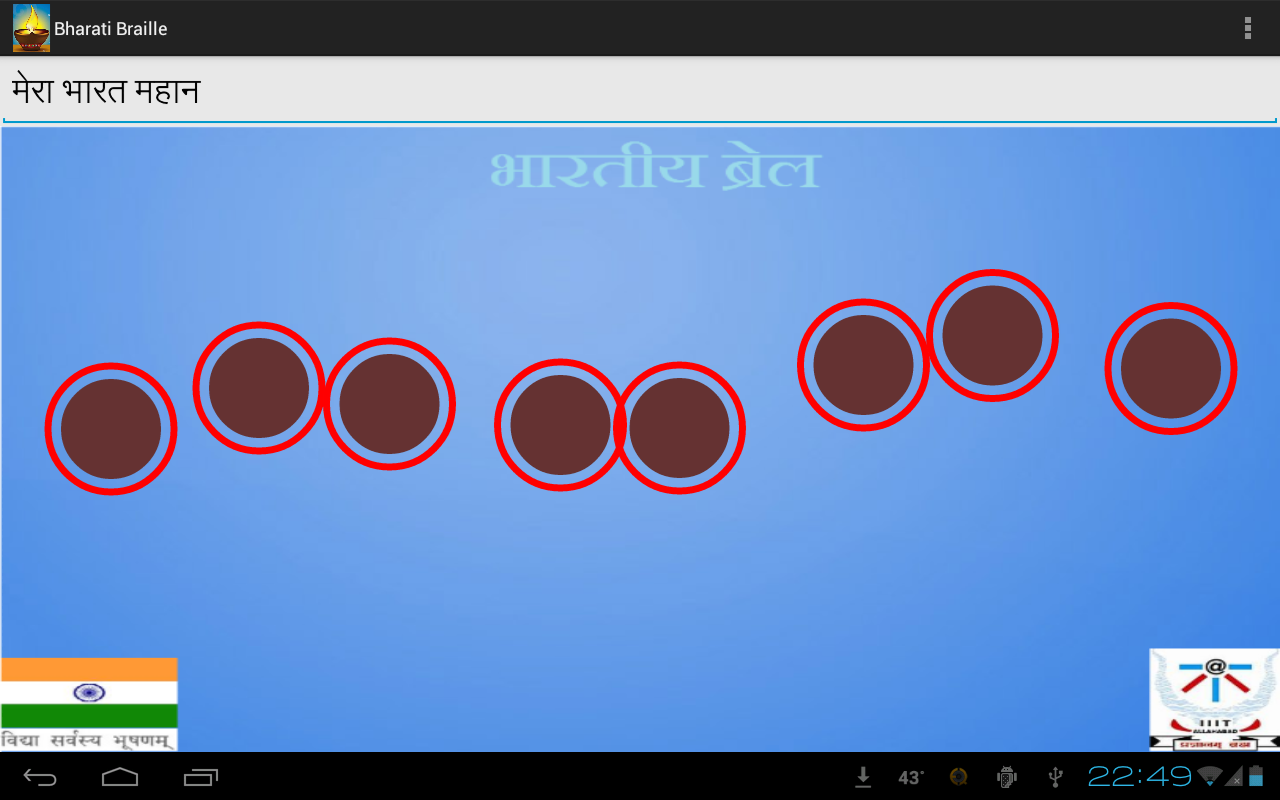




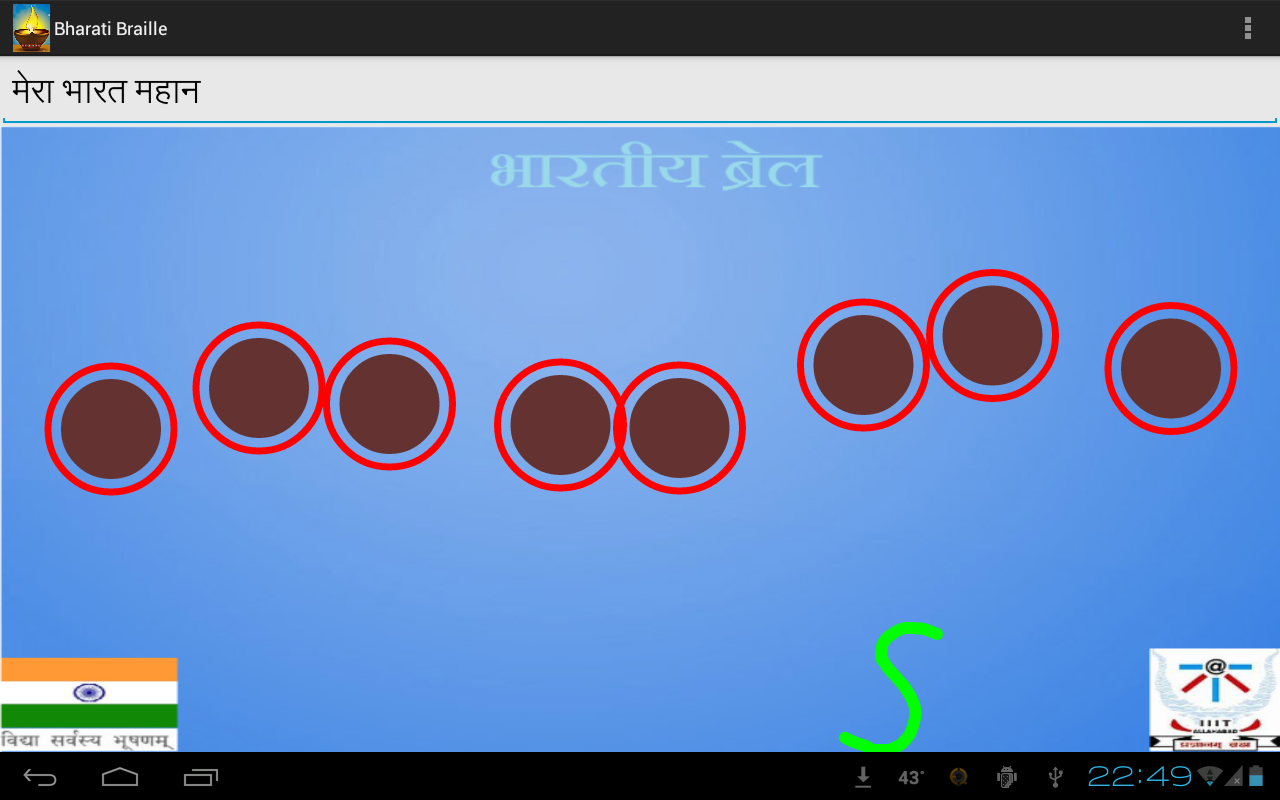


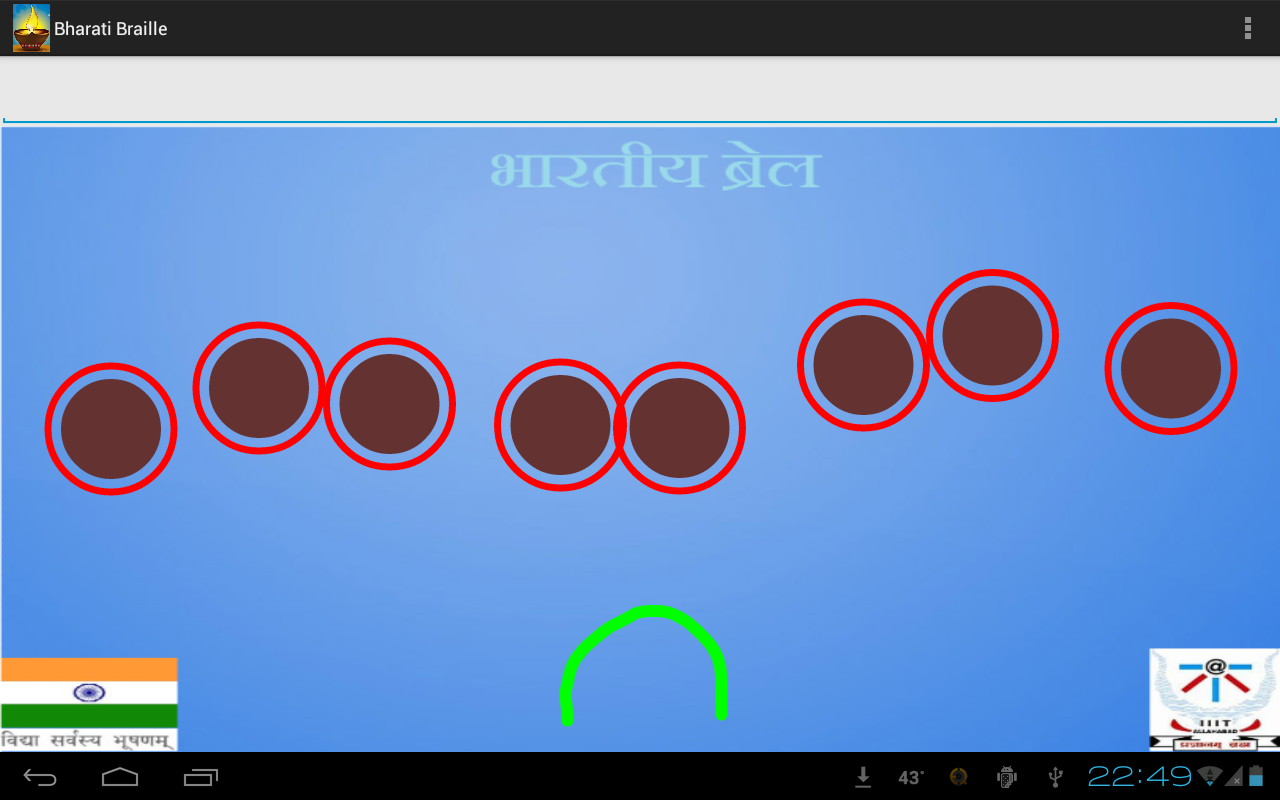


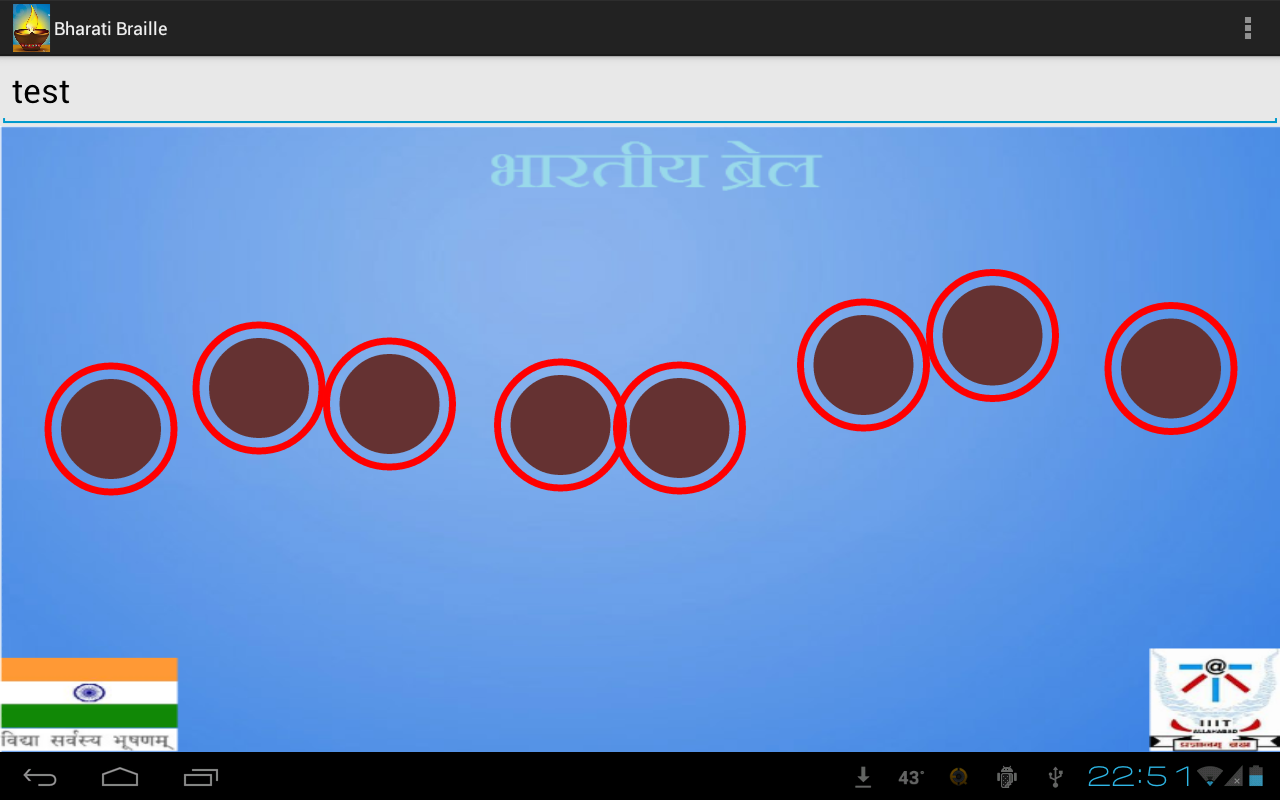




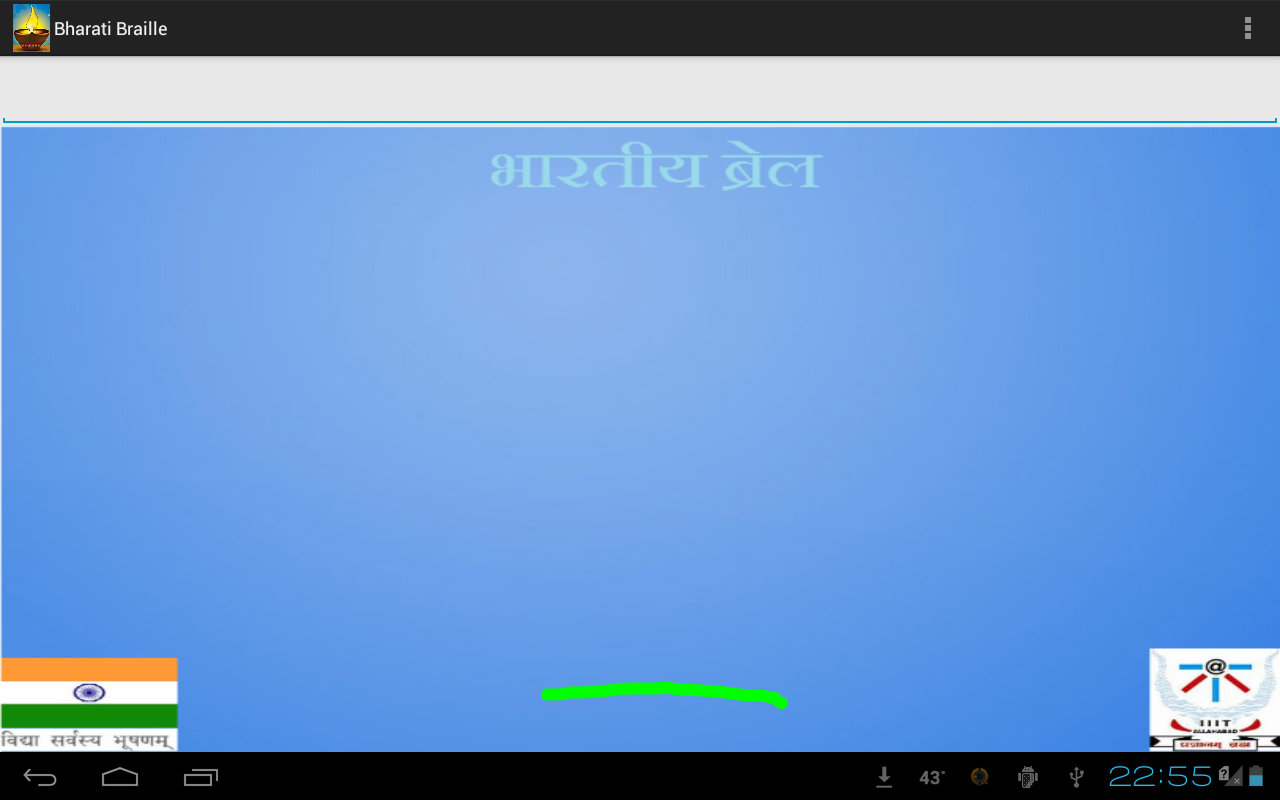
After entering the sentence, to save, we just need to draw ‘**S**’ and then enter the name for the file and then again draw an ‘**S**’ to save it. Similarly we can retrieve the file just by drawing a “**Straight Line**”(either forward or backward) on the bottom of the screen.

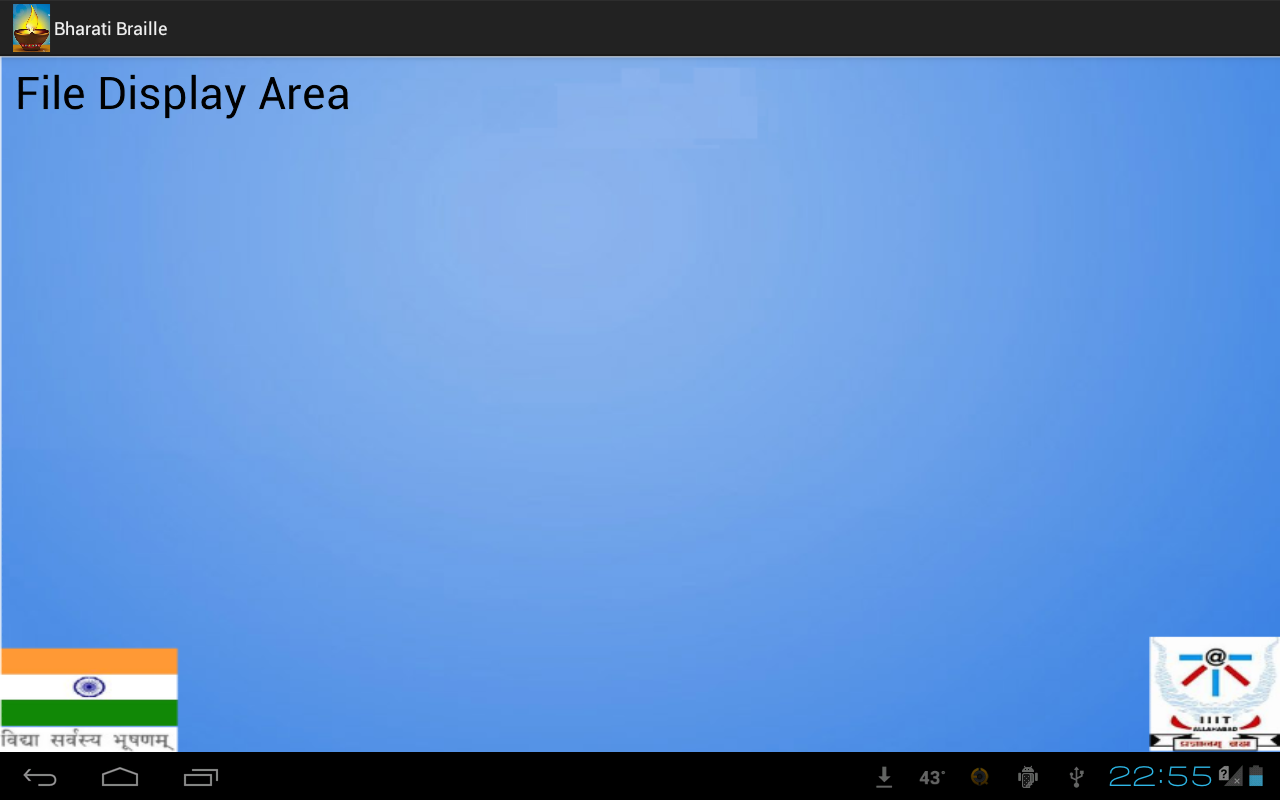


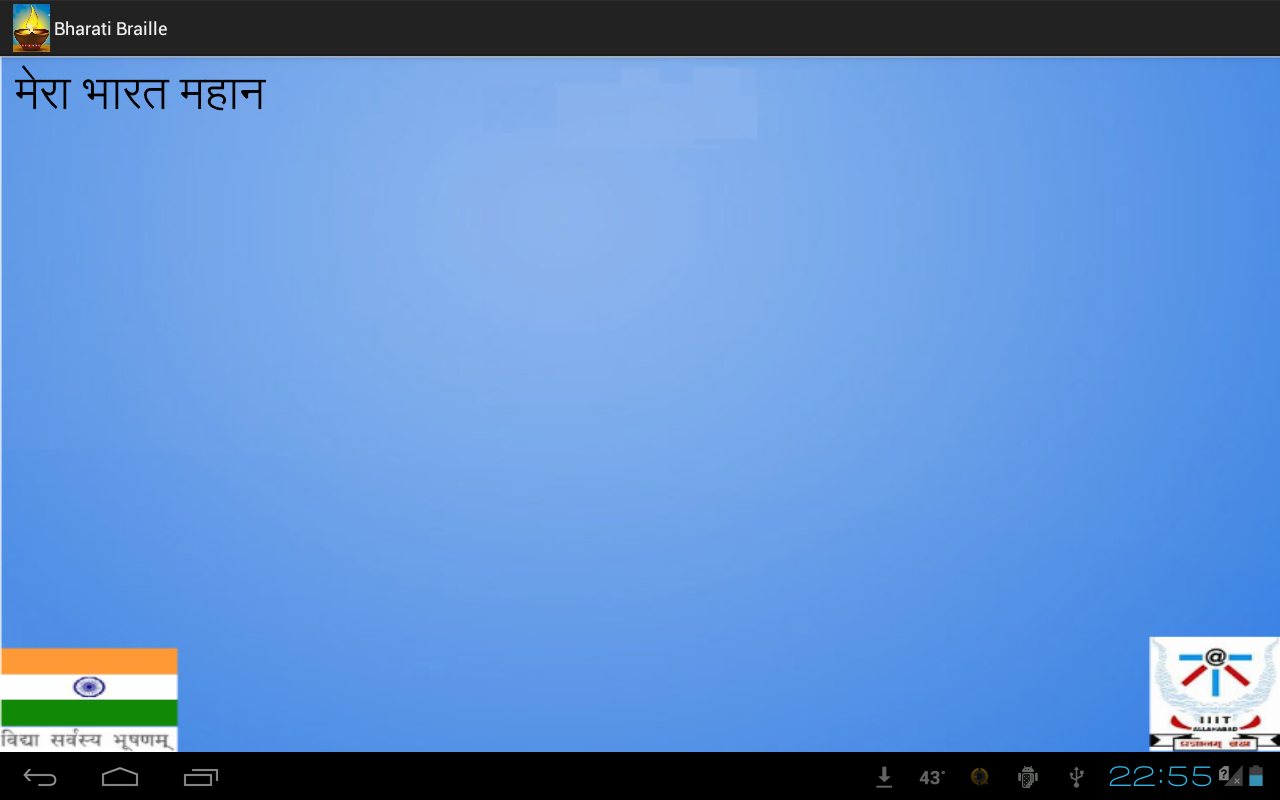








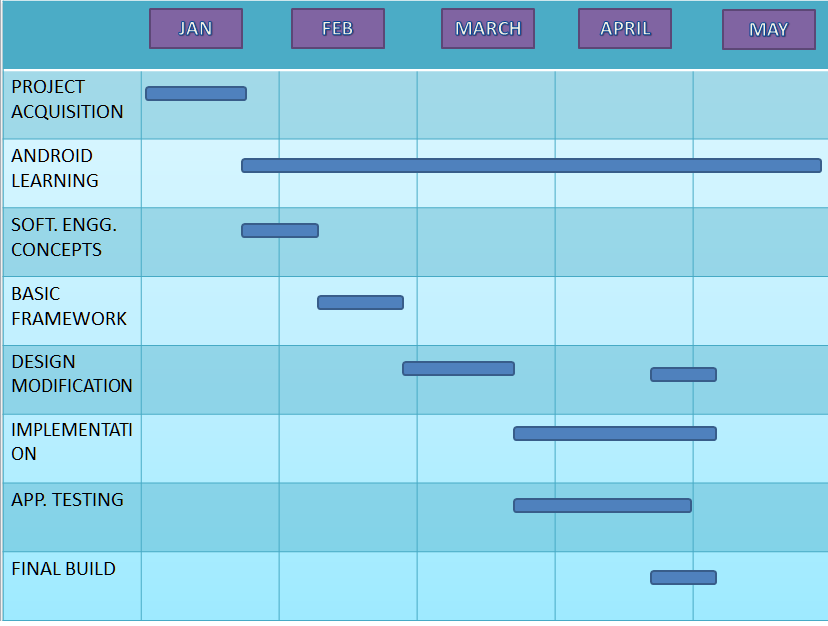




5.2 Analysis

Application has been successfully tested and verified against all marginal test cases as well as for general cases. We have not fully functional application capable of accepting and saving user data.

**6.Activity Time Chart**

****Fig.6.1 showing out timeline for the development of application.

Android learning stands for, learning the basic framework and language syntax to work with it

Software Engineering concepts were revisited to make sure everything is going right.

Testing and Implementation were done side by side to make sure all the critical test cases are covered during development.

**7.Conclusion**

This project was undertaken to implement a Hindi Braille Simulator on Android Platform which is quite efficient in terms of cost and portability as compared to the bulky Braille Typewriters. Apart from this our main motive was to contribute to the society by helping visually impaired people to express their thoughts in the form of written expression.

We can easily conclude from the above given demonstration that our application is fully functional and satisfies all the basic requirements that were needed.

**8.Recommendations and Future Work**

Our Project is based on Component Based Software Engineering(CBSE) model so new features can be easily integrated into the application without affecting the other functionality or the code.

Future work may include a more comprehensive and intelligent phonetic support, so that one may type in any language and that can be converted into any other language. This would eliminate the need for remembering large bit combinations and would greatly reduce the difficulty level.

We Recommend that if someone wants to take this work forward, then more emphasis should be given to language conversions.

**References:**

[1]Stanford Summer of Code

<http://engineering.stanford.edu/news/stanford-summer-course-yields-touchscreen-braille-writer>

[2]https://play.google.com/store/apps/details?id=kr.goodie.braille&feature=search\_result

[3]Android application

https://play.google.com/store/apps/details?id=com.nikolakirev.braillewriter&feature=search\_result

[4]English Braille Explanation

<http://en.wikipedia.org/wiki/Louis_Braille>

[5]UNESCO World Braille Usage

http://www.duxburysystems.com/documentation/manuals.asp?book=World\_1990

[6]Wikipedia Page for Bharati Braille

<http://en.wikipedia.org/wiki/Bharati_Braille>

[7]Schaum's Outline of Software Engineering (book).