CCNY CSc 221 Software Design Lab
Final Android Project Final Report, 20121220

David Prager Branner

Friday 21st December, 2012

1 Overview

This is a game, in prototype, to help the user develop speed in the recognition of Chinese characters. The user is shown a "target" character and must select it from among a screenful of "decoy" characters.

The goal of the game is for the user to improve his speed at selecting the target from among the decoys. This skill is directly applicable to proficiency in reading Chinese, and also to the most common task now used in generating Chinese text on a phone or computer.

Note: the present report describes a later version of the project than my presentation slides of 20121218

2 Responsibilities

I did this project entirely alone, consulting only existing on-line forum-discussions (primarily Stack Overflow) until 20 December, 2012, when I paid a half-hour visit to the instructor and asked help with two issues.

3 Detailed Description of Components

1. `MainActivity.java`: Opening screen.
   (a) Presents a description of the game.
   (b) Presents a button to take the user to a settings screen in `Settings`.
   (c) This page finishes and cannot be revisited after the user leaves it.
2. `Settings.java`: Settings.
   (a) Uses the `NumberPicker` widget and a button to let the user set the number of columns and rows in `ShowDecoyTable`.

3. `ShowDecoyTable.java`: The main screen of the game
   (a) Presents a grid of characters, all decoys except for one target.
   (b) Below the grid is a navigation bar, containing
      i. a button to return to `Settings` to change settings;
      ii. a colored `TextView` containing the target in the same font size as the grid;
      iii. a button to generate a new grid by.
   (c) Each character in the grid has an `OnClickListener` that colors the character when the user touches it: blue if the character is indeed the target, red on black if the character is not the target.
   (d) This screen is composed of a vertical `LinearLayout` holding two child-components:
      i. a multi-row `TableLayout` to hold the grid;
      ii. a single row `TableLayout` to hold the navigation bar.
   (e) Within the multi-row `TableLayout` of the grid, each cell contains a `TextView` holding a Chinese character and `OnClickListener` (described above).
   (f) Characters are supplied by calls to the `GenerateKanji` class.
4. `GenerateKanji.java`: Class to generate random characters and select a target from among them.
   (a) `generateRandKanji()`: Generate a random Chinese character. Procedure:
      i. select a random decimal integer within the appropriate range;
      ii. convert it to a `char` using `codepointToChar`.
   (b) `codepointToChar`: Convert a decimal codepoint to a Unicode character. Procedure:
      i. convert the decimal codepoint to a hex string;
      ii. prepend the \u Unicode prefix;
      iii. convert the resulting Unicode string to a `char` (required special trick).

   Code:

```
1     public char codepointToChar(int iCodepoint){
2         // Given a decimal codepoint, return a char containing the kanji.
3
4         // 1. convert to hex string (http://stackoverflow.com/a/516614; accessed 20121113)
5         // and then append UTF prefix
6         String uCodepoint = "\\u" + Integer.toHexString( iCodepoint );
7
8         // 2. "convert" to char (http://stackoverflow.com/a/2126394 ; accessed 20121113)
9         // The following fails because \\u is not recognized as the UTF prefix
10        //     char theKanji = uCodepoint.toCharArray()[0];
11        // Instead, do more complex substring process
12        // (http://stackoverflow.com/a/2126404 ; accessed 20121113)
13        char theKanji = (char) Integer.parseInt( uCodepoint.substring(2), 16 );
14        return theKanji;
15    }
```

   (c) `fillArray(int)`: Populate an array of random characters, assigning the `target` and `targetIndex` variables in the process. Cardinality of array is `int`.

4 Challenges

4.1 Choice of layout

It was difficult to know at the outset which layout would best serve this project. All the layouts have restrictions of different sorts, and at the moment I know of no comprehensive checklist or flowchart to help consider them before the fact.

My main choice was among `GridView`, `GridLayout` and `TableLayout`. There was a long struggle to compare them, but the vital issues were as follows:

1. Cell-to-cell spacing. In `GridView` it is difficult to get good control of the spaces between cells dynamically, and I eventually abandoned it. `GridLayout` and `TableLayout` have better control of cell-spacing.
2. Scrolling. `TableLayout` does not scroll, but I decided against scrolling, to keep things simple. When embedding `GridLayout` in `ScrollView`, the scrolling is slow and jerky.
3. Selection of individual cell contents. `TableLayout` does not directly support give per-cell selection, but I got around this by placing a separate `TextView` in each cell. The downside of doing that is that when the dimensions of the grid are large, its population and display are slow. `GridLayout` allows cell-level selection, but that requires writing custom adapters — considerably more work than what I ended up doing with `TableLayout`.

There were two other challenges that arose as the result of choosing `TableLayout`.

4.2 Dynamic layout

Normally, the dimensions of both `GridLayout` and `TableLayout` are set in `Main_Activity.xml`. Since I intended for the dimensions to be set by the user in all situations, it was simpler to instantiate my layouts programmatically.[1] Not all attributes can be set programmatically, however, and in some cases programmatic access requires a certain API level.

4.3 Font size

The `TextViews` I seated within `TableLayout` cells did not adjust their font-size optimally, so I wrote a code block to accomplish the same thing using an actual measurement of screen width:

```
1     float fontsize;
2     // In order to decide font size, first find screen width,
3     // (see http://stackoverflow.com/a/1016941/621762)
4     // then divide by (approx.) number of columns.
5     Point screenDimensions = new Point();
6     Display display = getWindowManager().getDefaultDisplay();
```

[1] The Android Developer site's arguments about the importance of segregating layout from content are somewhat ideological; in some respects — cognitively, for instance — content and layout are not distinguished in the superficial way they may seem to our conscious minds. It is best for the programmer to decide what the logical subdivisions of his project are. Requiring XML means a second syntactic system in a project and more windows visible at a given time, which lead to greater complexity in managing the coding process. Even in a project without XML, the programmer can segregate content-bearing fields from rest of his code, so the utility of an XML component seems to me overstated.

```
7     display.getSize(screenDimensions);
8     // Font size in pixels - this is more useful than sp.
9     // We use cols + 1 in order to force room for an extra row at bottom.
10    fontSize = screenDimensions.x/((float)cols+1);
```

4.4 Testing

Hardware and software involved:

```
Mac OS 10.8.2 on 11-inch MacBook Air, Late 2010
Eclipse Version: Juno Service Release 1
Build id: 20121004-1855
Android tools: 21.0.0.v201210310015-519525
```

I tried using emulators first. Of the three CPU/ABI options, only MIPS seems to load successfully, and I used the 3.7-inch FWVGA slider.[2] Physical devices seem to need to be disconnected in order to use emulator. Sometimes there is a long wait after console message

```
Waiting for HOME ('android.process.acore') to be launched...
```

and following `http://stackoverflow.com/a/4968923/621762` running the program a second time (without killing the first instance. is more successful. Emulator must be unlocked by dragging lock icon to right.

Since the emulator was slow, and I did most of my work on physical machines. I used two HTC phones, Android OS v. 2.3.4 (API 10) and v. 4.0.3 (API 15). Because some of the tools I tried required an API of at least 14, I ended up using the API-15 machine for most of my work.

4.5 Failure to "finish"

When `finish()` is called after `startActivity()`, if the new activity takes a long time to load and if a third activity is immediately called from the second, it is possible that the original `finish()` statement will not be completed. This can go on in a series of activities and slow the phone down substantially. I think there are ways, from within a new activity, to ensure that a prior activity does indeed finish, but I have not looked into them yet.

4.6 A scope problem connected with inner classes and intents

Anonymous inner classes such as `onClick()` require that any variables declared in the outer method be `final`, and this creates a problem if those variables need to be modified in the outer method. They can be declared as class fields — before any method declarations — and that makes them globally accessible and eliminates the need for them to be made `final`.

[2] No keyboard was apparent on this emulator, despite "slider" in the name. For user-keyboard-to-Emulator-keyboard mappings, see `http://developer.android.com/tools/help/emulator.html` (accessed 20121215).

However, there is at least one sort of field that cannot be declared this way: one that is passed as a `putExtra()` arguments in an intent from another activity. Passing an intent seems to require context to have been established, and that means that `putExtra()` arguments cannot be passed (or at least received) until after `onCreate()` has been called, and at that point `final` seems to be required.

To avoid the use of final, declare variables that will receive `putExtra()` arguments as class fields, without `final`, but actually assign them within `onCreate()`. Here is an example from my code involving the variable `cols`:

```
public class ShowDecoyTable extends Activity {
    int cols; // as a class field, does not need to be final
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        cols = this.getIntent().getIntExtra("cols", 7); // already declared
        ...
        Button repopulateButton = new Button(this);
        repopulateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(ShowDecoyTable.this, ShowDecoyTable.class);
                intent.putExtra("cols", cols); // available to onClick()
                startActivity(intent);
                finish();
            }
        ...
    }
    ...
}
```

5 Chinese Characters

5.1 Generation of characters

Eventually, I intend to use a database to select characters randomly and for pedagogical value. But for the time being I chose a simpler method:

I generate Chinese characters randomly by constructing `char` values consisting of the Unicode prefix `\u` followed by a four-digit hex value in two codeplanes

1. most common characters: CJK Unified Ideographs (hereafter, “Unified Ideographs”; codepoints 4e00–9fff); both modern traditional and simplified forms are used;
2. rarer characters: CJK Unified Ideographs Ext. A (hereafter, “Ext. A”; codepoints 3400–4dff); some of these are quite obscure to modern readers — hence their utility as decoys.

I have omitted other CJK codeplanes as too recondite for present purposes

1. CJK compatibility Ideographs (f900–faff); these are variant commoner in historical texts than the Unified Ideographs or Ext. A, and also tend to appear in Japanese and Korean usage.

2. Kangxi Radicals (codepoints 2f00–2fdf)
3. CJK Radicals Supplement (codepoints 2e80–2eff);
4. CJK Strokes (codepoints 31c0–31ef);

or too computationally difficult because surrogate pairs would be involved:

1. CJK Unified Ideographs Ext. B (codepoints 20000–2a6df);
2. CJK Unified Ideographs Ext. C (codepoints 2a700–2b73f);
3. CJK Unified Ideographs Ext. D (codepoints 2b740–2b810);
4. CJK Compatibility Ideographs Supp. (2codepoints f800–2fa1f).

Some rather rare characters, as well as independent combining-form radicals, also appear in the regular Unified Ideographs and Ext. A blocks. Not until I incorporate a database into this project will it be easy to prevent them from coming up at random.

5.2 Which characters can appear on an Android machine without worrying about fonts?

I prepared a list of the characters in the CJK Unified Ideographs Unicode block: `\u4300` to `\u0fc6` and pasted them into an Evernote document, to see which, if any, failed to appear. Those numbered `\u9fb4` to `\u9fc6` (decimal 40884–40902) failed to appear, so I have removed them from my set of possible characters.

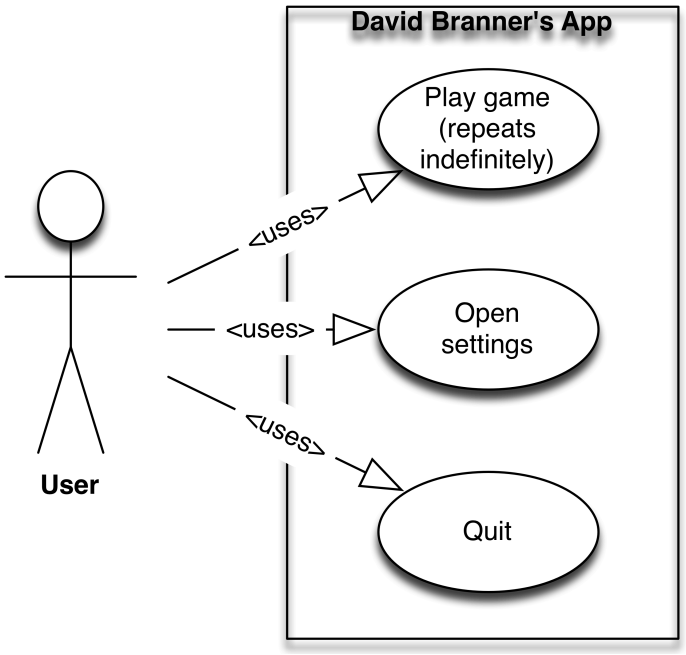
In the block of those somewhat rare, `\u4db6` to `\u4dff` (decimal 19894–19967) failed to appear, so I have removed those, too.

The remaining code blocks in the CJK Unified Ideographs and Ext. A all seem to be present in the font on my two test machines, and I have experimented for a while to ensure that no undisplayable characters are generated. I think a bit more work is needed.

6 Formal representations of structure

6.1 UML Use Case Diagram

No change from the original high-level use cases:



6.2 UML Class Diagrams

Note: anonymous inner classes are not shown, but their fields are.

1.

MainActivity
theLayout : LinearLayout theView : TextView settingsButton : Button intent : Intent menu : Menu
onCreate(Bundle) : void onCreateOptionsMenu(Menu) : boolean

2.

SetSettings
theLayout : LinearLayout theView : TextView theNumberPicker : NumberPicker theNumberButton : Button pickerHeight : int numbPickrAndButton : LinearLayout intent : Intent cols : int i : int minCol : int maxCol : int v : View
onCreate(Bundle) : void

3.

ShowDecoyTable
col_i : int row_i : int index : int fontSize : float targetIndex : int rows : int g : GenerateKanji kanji : String[] target : String actionBar : ActionBar theLayout : LinearLayout theDecoyTable : TableLayout screenDimensions : Point display : Display theRow : TableRow theView : TextView v : View indexClicked : int theNavigationTable : TableLayout theBottomRow : TableRow rowsButton : Button theTargetView : TextView repopulateButton : Button intent : Intent menu : Menu
onCreate(Bundle) : void onCreateOptionsMenu(Menu) : boolean

4.

GenerateKanji
cols : int rows : int col_i : int row_i : int index : int generator : Random target : String targetIndex : int kanji : String[] clicked ; String numbKanji : int min : int man : int randCodepoint : int iCodepoint : int uCodepoint : String theKanji : char currentKanji : char
generateRandKanji() : char codepointToChar(int) : char fillArray(int) : String[]

[end]

7 Original proposal follows

7.1 Summary Description

7.1.1 Nature and goal

This is a game to help the user develop speed in the recognition of Chinese characters. The user is shown a "target" character and must select it from among a screenful of "decoy" characters.

The goal of the game is for the user to improve his speed at selecting the target from among the decoys. This skill is directly applicable to proficiency in reading Chinese, and also to the most common task now used in generating Chinese text on a phone or computer.

This project is intended to be a proof of concept. Features that may be added later to make the task more challenging, should there be time and opportunity, include:

1. Gradually reducing character size.
2. Increasingly obfuscating the characters in various ways.
3. Populating the set of decoys with those that share components with the target. For instance, if the target is 銀, incorrect characters could include
   - those containing the element 金 : for instance, 釗, 鈐, 鈑, 鈸, 鈺, 鈔, 鈕, 鈖, 銃, etc.
   - those containing the element 艮: for instance, 很, 眼, 根, 跟, 限, 退, 恨, 艱, 狠, etc.

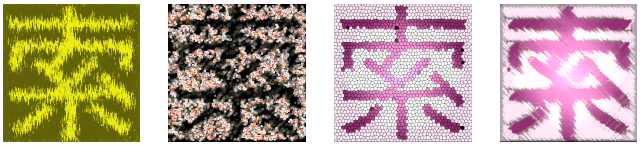
   This will require greater skill at recognizing *whole* characters, an important step in developing reading speed in Chinese.
4. For more advanced users: Giving the user not the actual target character but some associated value, such as its pronunciation, etc. However, this feature introduces myriad problems of standardization.
5. Choice of target to be selected from graded official list of Chinese characters (there are a number of such lists to choose from).

7.1.2 Possible names

- "zFnd": Ugly but distinctive; suggests "find *zì* [Chinese characters]".
- "Suozi" *suǒzì* 索字 [look for characters]. Pronounceable, (though likely to be mispronounced by people who do not know Chinese).

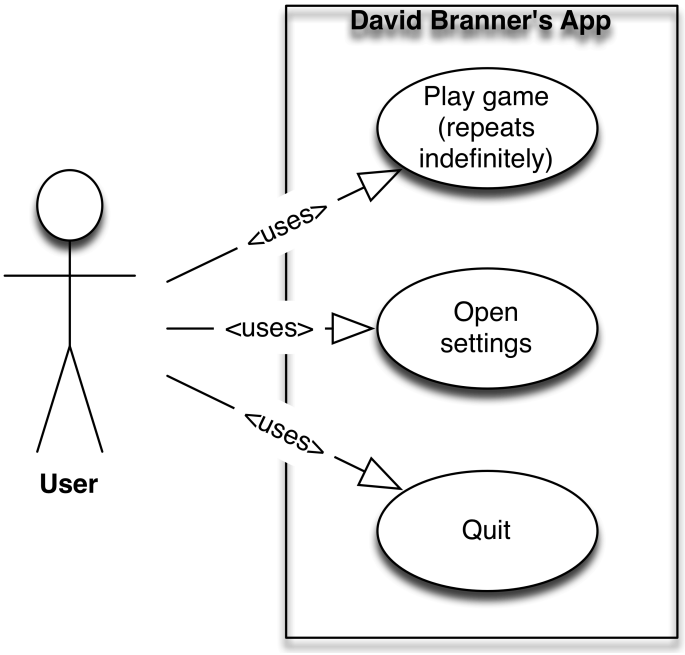
7.1.3 Possible app icons

To consist of the graph 索 'look for', partially obfuscated:

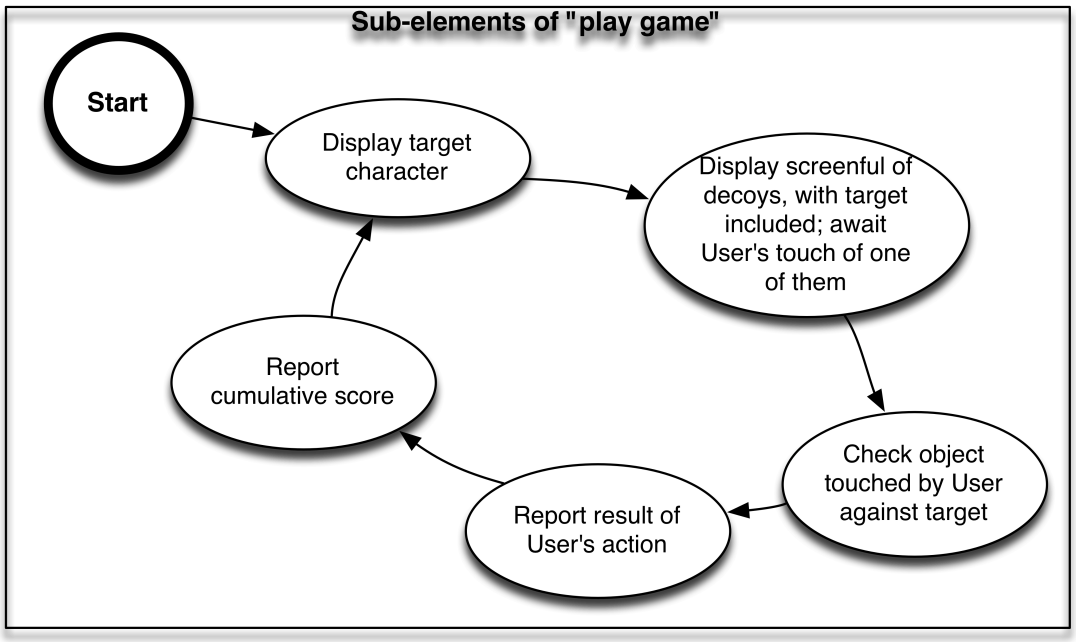


7.2 UML Use Case Diagram

7.2.1 High level use cases



7.2.2 Details of "play game"



7.3 Detailed Description of Components

1. `GenerateTzyh`: Class to perform functions involving Chinese characters.

   Fields:

   (a) `generator : Random`: A random number generator.

   (b) `target : char`: The target character.

   Methods:

   (a) `avoidSpecialTzyh(char)`: Generate random characters until the specified one is reached.

   (b) `codepoint2Char(int)`: Given any decimal Unicode codepoint, return a char containing the corresponding encoded Chinese character.

   (c) `generateRandTzyh()`: Generate a random integer and from it call codepoint2Char.

   (d) `checkIfMatch(char)`: Report on whether the User has correctly identified the target.

   (e) *for later*: Methods to restrict target to a particular subset of characters.

   (f) Note: high Unicode codepoints, requiring the use of surrogate pairs, are apparently not support on Android by default.

2. `Decoys`: Class to populate an ArrayList with one instance of the target and the rest "decoys" (random Chinese characters)

   (a) Method to ensure that no decoy is the same as the target.

   (b) *for later*: Methods to refine the choice of decoys to increase difficulty.

3. Settings class
   (a) `displaySize`: Number of characters per "screenful"; will we use a square grid? if so, this value should be number of characters per side.
   (b) `score`: For future use in scoring analytics, this will be modeled as an ArrayList.
   (c) *for later*: other settings related to added difficulty (obfuscation, distractions, etc.) and to choice of characters for graded self-testing.
4. Class to manage Views
   (a) Grid of characters
   (b) Settings
   (c) Display of target
   (d) Report of results of each round of the game
   (e) Report of running score

7.4 UML Class Diagrams

1.

CharacterGenerator
target : char generator : Random
avoidSpecialTzyh(char) : void checkIfMatch(char) : void codepoint2Char(int) : char generateRandTzyh() : char

2.

Decoys
[pending — perhaps only a single method]

3.

Settings
displaySize : int score : ArrayList
[pending]

4.

ViewManager
[pending]

7.5 Group Responsibilities

I have decided to do this project alone.