24 Game

COMP7506 Smart Phone Apps Development
Assignment 1

Yuteng Zhong
MSc (CompSc)
3035169867

1. Introduction

The 24 Game is an arithmetical card game in which the objective is to find a way to manipulate four integers so that the end result is 24. Addition, subtraction, multiplication or division, and sometimes other operations, may be used to make four digits from one to nine equal 24. For example, card with the number 4, 7, 8, 8, a possible solution is the following (7 - (8 / 8)) * 4 = 24. [1]

In this assignment, we expand the digits from one to thirteen, including Jack, Queue and King.

2. Basic design

According to the assignment specification, the application should randomly pick 4 non-duplicate cards from totally 52 poker cards when game starts. User can choose the selected cards and arithmetic symbols displaying on the screen. [2]

So my project consists of three major parts:

1. Card dealer, which is for randomly pick 4 non-duplicated cards and ensures the picked card set has at least one solution.
2. Equation evaluator, which is for evaluate the formula that user inputs. And it also contains a formula validator to point out user input error.
3. Score manager, which is for managing and storing the scores.

These three modules should be in 3 different classes.

3. Implementation details

3.1. User Interface

According to the specification, the application should at least consist of two Activities:

1. A welcome Activity, which displays candidate's name and university number for identification purpose. And this page should provide a button for entering the game.
2. Main Activity, which contains all gaming logics. There will be:
   a. 4 Buttons for representing 4 poker cards,
   b. 6 Buttons for representing "+", "-", "*", "/", "(" and ")" operators,
   c. 2 TextViews for displaying formula and result
   d. 1 Start new game Button
   e. 1 backspace Button
   f. 1 clear Button
   g. 1 evaluating Button, which should be represented by "=" sign. If pressed, the application should evaluate the formula that user inputs and give the result or error messages. When it is pressed again, new game starts if play wins, or the formula will be cleared to let the player to play again.

In my implementation, I have combined the two TextViews into one, and combined the clear button and the backspace button into one. Just for better looking and implementation convenience.

The Splash Screen

The welcome screen is called "Splash Activity" in the application, which is implemented in SplashActivity.java, and it is the first screen we will see when we launch the application. As we can see in Figure 1, the root layout of the

Activity is a RelativeLayout. The poker card of “24” is a ImageView, and the information of myself is in the ImageView on the right bottom of the screen.

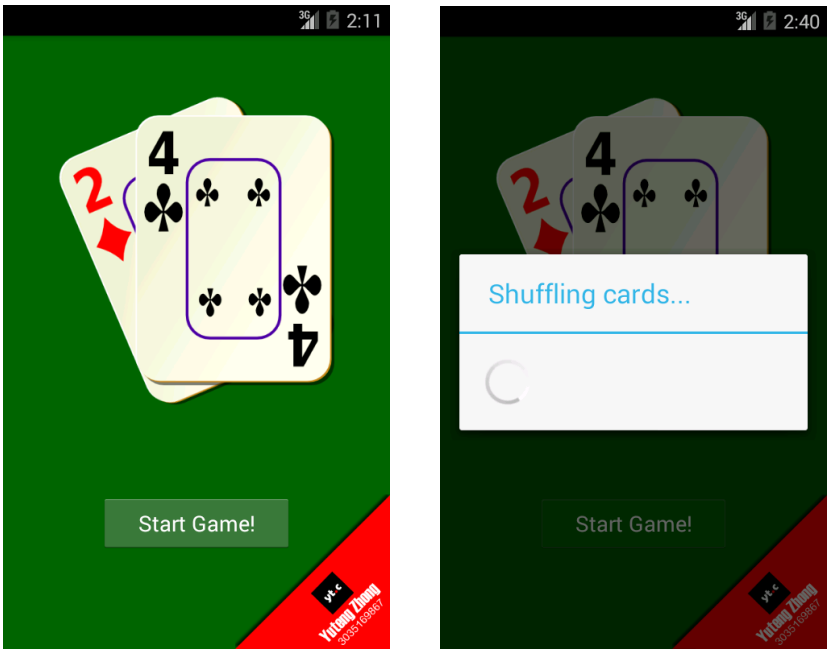

Figure 1: Splash Activity

When the “Start Game!” button is press, it will lead us to the “Game Activity”, the main Activity of the application. As the “Game Activity” will take a little time to construct, a ProgressDialog with title “Shuffling cards …” will be shown until the “Game Activity” is loaded.

The layout of the Activity is defined in activity_splash.xml.

The Game Activity

The “Game Activity” is implemented in the GameActivity.java. When entering the Activity for the first time, application will first generate a new card set. Because validating the card set may take a long time (enumerate all possible solutions), a ProgressDialog will be shown until one valid card set is found.

When the valid card set is found, cards will be set as Selectors of the four ImageButtons in the centre. Then an animation will be performed, which will

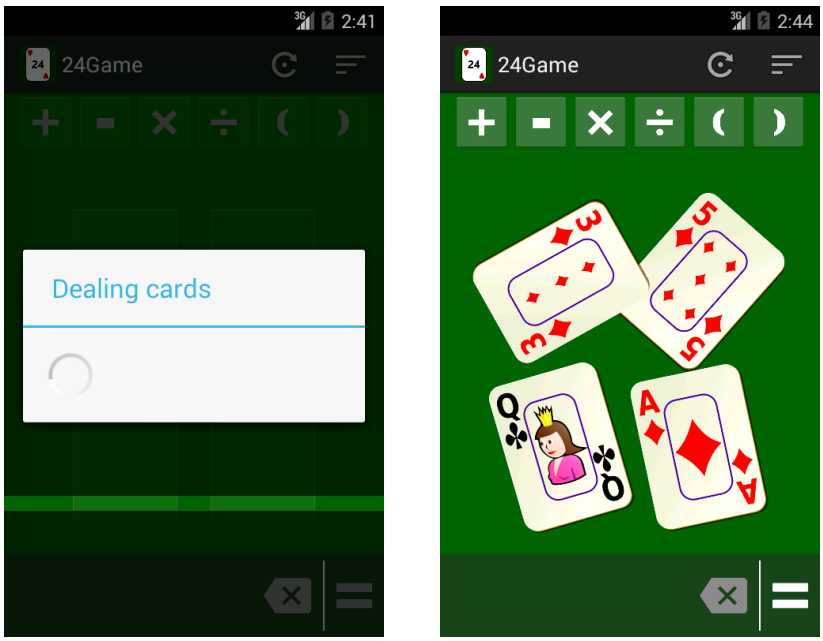

Figure 2: Entering the Game Activity

move the four cards from upper right to the centre of the screen with rotation. It looks like dealing cards in real life.

After dealing cards, all button will be unfreeze. The two MenuItems on the ActionBar are for restart the game and open the “Score board Activity”. When pressing the restart button, the game will be restarted. The operator buttons on the upper edge of the screen, the four cards in the centre of the screen will

Figure 3: Input and error messages

become the operators and operands of the formula. When pressing them, the specified formula components will be added into a buffer and shown as a formula in the TextView at the bottom. A validation process will perform when user is inputing, so that the user can be informed in realtime. The error messages will be shown as Toasts.

Figure 4: Evaluate the formula

The TextView at the bottom is for showing the inputed formula and the evaluation result. The two buttons on the right are backspace button and evaluate button. When pressing the backspace, it will pop the last inputed formula component. If no input left after pressing, the backspace button will be disabled. The evaluating button is for calculating and validating the value of the equation. Errors will be shown as Toasts during evaluation. If the value of the equation is 24, than the application will append “=24” into the TextView, otherwise it will append “≠24” and set the font color to red.

The four buttons in the centre represent four poker cards. When pressed, it will become disabled immediately (showing the back of the card). If the input

is popped by user, the button will be enable again. The angles of the buttons will be set randomly every time when the game starts.

Figure 5: Score board Activity

The Activity's layout is defined in activity_game.xml.

The Score board Activity

The "Score board Activity" is for showing the statistics of the user's wins and loses, also including the recent 10 game results.
The "Score board Activity" is implemented in the ScoreBoardActivity.java. You can enter this activity by clicking the upper right button in the "Game Activity".

The root layout of the Activity is a LinearLayout and it has only one child, a ListView. The upper part of the screen is the list header, which is defined in listheader_score_board_statistic.xml. The numbers indicates how money rounds did the user won or lost. The layout of the list item is defined in listitem_score_board_scores.xml. The items from left to right are the row id, time in seconds that user have used, "✔" means wins and "✖" indicates loses.

3.2. Utilities

The card dealer

The card dealer in the application is responsible to generate 4 cards randomly and ensure it has at least one solution. All implementation details are located in the CardDealer.java.

Deal to the limitation of memory on the portable devices, it is unpractical to load all 52 cards' image data into the memory. So I implement a LRU cache to store the recently used image data.

All the cards are stored as a ArrayList in the memory. I use a simple sampling algorithm to pick 4 cards from the card set. Then I pass the selected cards into a validator (GameSolver.java). If the validation not pass, then the process will be repeated until a valid card set is found.

The game solver

The game solver is a validator for validating the given card set has at least one solution. The validation method is simple and brute, which is to enumerate all possible solutions and see if there is at least one solution could fulfill the requirements.

For one of the permutations of the card set, such as [1, 2, 3, 4], the algorithm will try to apply all possible operator permutations on it. For example,

Card permutation: [1, 2, 3, 4]
Operator permutation: [+, *, -]
Possible solutions:

(1 + 2) * 3 - 4
1 + (2 * 3) - 4
1 + 2 * (3 - 4)
((1 + 2) * 3) - 4
(1 + 2) * (3 - 4)

...

We could realize that the problem could be solved by recursion steps.

1. If the expression is empty, the value should be zero
2. If the expression contains only one number, the value should equal to the number, such as “1”
3. If the expression contains more than one number,
    1. Try to add brackets to embrace every two numbers and a operator, such as “1 + 2 * 3” could be processed as “(1 + 2) * 3” and “1 + (2 * 3)”
    2. Evaluate the expression between the brackets, the result is E. And then evaluate the left and the right part of the expression recursively and get the results L and R. Then evaluate the expression “L op1 E op2 R”. It is noted that the “L op1 E op2 R” can also be evaluated in two ways: “(L op1 E) op2 R” and “L op1 (E op2 R)”.

So, for every permutation of the card set, test all possible permutations of operators and see if there are results equal to 24.

Because in this game, the number of operands must be 4. So the number of operators have to be 3. So it will not be necessary for the algorithm to generate all permutations of operators every time. Another optimization is use next lexicographical permutation algorithm to iterate all permutations of cards without generate all of them in advance, which will save lots of memory and time.

Another possible optimization of this algorithm is to avoid duplicate evaluation by using the operator priorities. But it is not implemented in this project.

The Equation

The Equation class in the project is to manage all the input operators and operands. It can also validate the user input and give error messages in realtime.

The operator and operand are subclasses of EquationNode, which is an abstract representation of the components in an equation. User's input will be converted into one of the subclasses of EquationNode and store into an array as buffer. The Equation has a state machine for ensuring the input is valid. The state machine could be described as Figure 6.

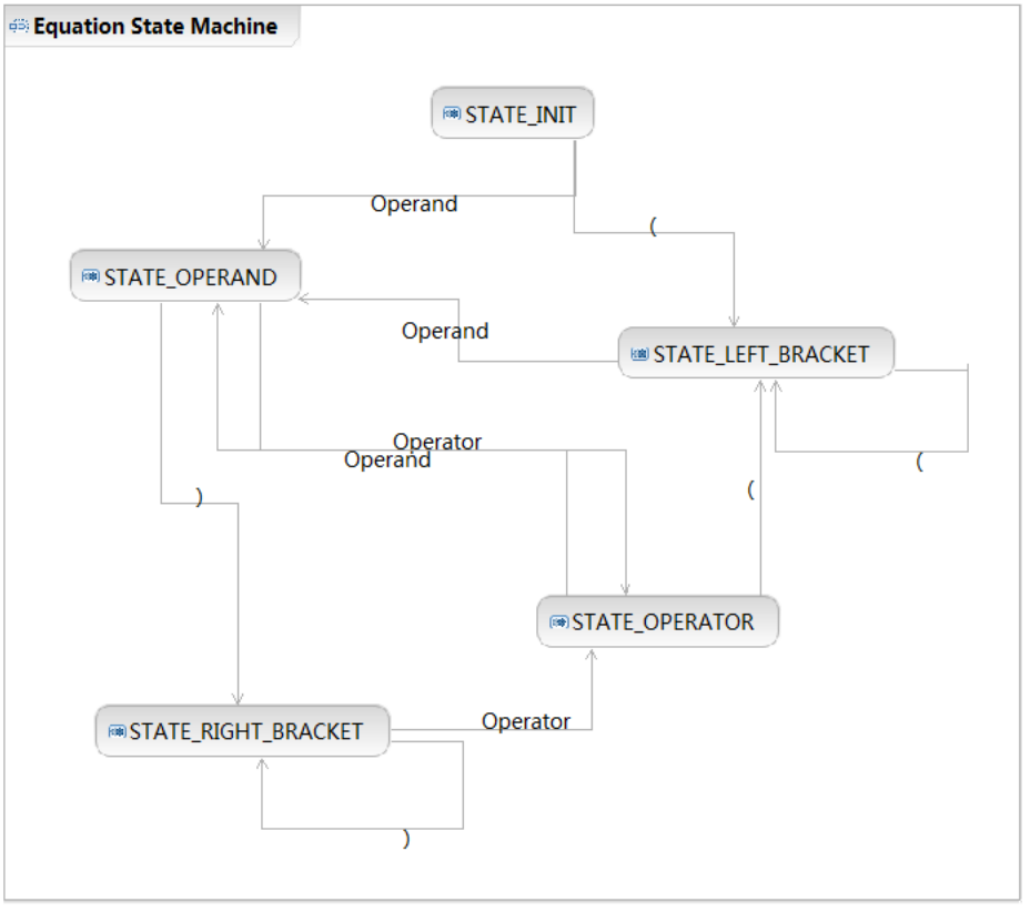

Figure 6: Equation State Machine

There are five valid states in the state machine. STATE_INIT is the begin state, which representing a empty equation. Adding or removing nodes may trigger state transition, so it is able to detect input error in time.

If the equation is now in a valid state, then user can call the evaluate() method to get the value of the expression. A valid equation should not end with an operator or a left bracket. The evaluation process could be described as:

1. Convert the existing equation from infix notation to postfix notation (Reverse Polish notation)
2. Evaluate the postfix expression and get the result

Infix notation is the most common arithmetic and logical formula notation, in which operators are written infix-style between the operands they act on [3], such as

$$5 + ((1 + 2) \times 4) - 3$$

Correspondingly, the postfix notation is a mathematical notation in which every operator follows all of its operands [4]. In the previous infix notation example, the expression in postfix notation is

$$5\ 1\ 2\ +\ 4\ \times\ +\ 3\ -$$

The conversion from infix notation to postfix notation algorithm is Shunting-yard algorithm. [5] This algorithm and the evaluation of postfix notation algorithm are in O(n) time complexity.

The Database

For remembering the user's historical score data, I use the Android's builtin SQLite database to store the time and the game result. The table schema should be

```
CREATE TABLE game24_score (
    _id INTEGER PRIMARY KEY AUTOINCREMENT,
    start_time DATETIME NOT NULL,
    end_time DATETIME NOT NULL,
    interval INTEGER NOT NULL,
    result INTEGER NOT NULL);
```

The “interval” column stores the time consuming of the user in this round, and the “result” column stores the game result, in which “1” indicates the user won and “2” indicates the user lost, other values should be considered as unknown game result.

4. Build the project

I use AndroidStudio to develop the application. AndroidStudio is a IntelliJ based Java IDE, which use Gradle as the build tool. The most simple and straightforward way to build and install a debug APK into the testing device is to execute the `gradlew` script under the root directory of the project. In Unix-like system, the command should be

```
./gradlew installDebug
```

The project is managed as the following tree

```
├── Game24.iml
├── README.md
├── app
│   ├── app.iml
│   ├── build
│   ├── build.gradle
│   └── src
│       └── main
│           ├── AndroidManifest.xml
│           ├── assets
│           ├── java
│           └── res
├── build
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── import-summary.txt
├── local.properties
└── settings.gradle
```

Figure 7: Project tree

The main part of the Android project is located in `app/src/main`.

If you want to build the project with Eclipse ADT, just new a project and import those files in `app/src/main`. The project should be built successfully.

References:

[1] Wikipedia, 24 Game, http://en.wikipedia.org/wiki/24_Game

[2] COMP7506 Smart Phone Apps Development Assignment 1 specification

[3] Wikipedia, Infix notation, http://en.wikipedia.org/wiki/Infix_notation

[4] Wikipedia, Reverse Polish notation, http://en.wikipedia.org/wiki/Reverse_Polish_notation

[5] Wikipedia, Shunting-yard algorithm, http://en.wikipedia.org/wiki/Shunting-yard_algorithm