COMP2422 Visual Interface and Interaction Development 2013 - 14

Assignment 1

The submission is divided into 4 parts, which are all included in the file.

a. This submission report (this file)

b. The UML sequence diagram (UML.png)

c. The prototype (hi-f_demo.jar)

d. The Java source code of the prototype (Flipper.java)

1. Usability Analysis

1.1 Effectiveness

The Flipper.java do not have high effectiveness. The goal of this applet is to judge the fairness of random generation in java. The applet only give a deviation value. To get the result (whether the generation is fair or not, or to which degree it is believed fair), user still need to analysis the deviation by himself, such as analysis how large the deviation is. As the applet do not complete its job, the effectiveness of this applet is still need to be improved.

Go to the implementation

1.2 Efficiency

Firstly, as specified in 1.1, the user should do some analysis by himself. It will be very time-consuming and cause inefficiency.

Secondly, if the user want to test a sequence of large values such as 1 billion, 2 billion and 3 billion(only approximation), the applet should run 6 billion times totally. On testing large values, it is also not efficient.

Go to the implementation

1.3 Function

As a statistical analysis, the result should be based on a large number of samples or trials. The applet only provide single trial each time and there is no function to keep a history record of trials before. So this applet do not have ability to do mass sample analysis. Thus, the results that users get from this applet may vary a lot. This can also improve the effectiveness of the applet.

Secondly, to get the direct result that the applet need to give, there need to be a mechanism to convert the deviation value into a Boolean value – fair or not. This will both improve the effectiveness and the efficiency of this applet.

Finally, the Flipper.java only test the random.nextBoolean method. The random generation of other data types are not covered. This is the incompleteness of its function.

Go to the implementation

1.4 Safety

The applet won't crash during operations. Also there's no related data security issues. The applet can be considered safe.

1.5 Learnability

The applet is of low learnability. Although the operations are simple because all operations are on only two buttons, the applet do not tell the user what it does and how it will achieve that. Also, a new user won't know what that three text fields and flipping numbers stand for.

Go to the implementation

1.6 Memorability

The applet is of high memorability. When the user have already learned how to use it, to memorize the functions with two button is simple

1.7 Accessibility

The applet is available online. After downloading it, it is a .jnlp file, which is accessible only under when the device is connected to the internet. The shortcoming of this applet is that, as a .jnlp file, it relies on the internet environment to run.

Go to the implementation

1.8 User Experience

The applet put everything horizontally, that will cause long distance movement of eye sight and cursor when the applet is being used. This will cause a little bit inconvenience.

Go to the implementation

2. Suggested Improvements

The high-fidelity prototype is attached with the submission, it is the hi-f_demo.jar, all the improvements without special comments below have been implemented.

2.1 improvement for effectiveness

In order to give the user a direct result on the generation, there need to be a threshold as a standard used to compare with the deviation. This is the "threshold" value. This value is user input. If the deviation is larger than this value, that means the generation is not fair. The color of the "fair rate" textfield will turn red. If the deviation is less than this value, the generation is considered fair and the corresponding color is green. The fair rate, as a cumulative result, will change after each generation (start/stop and pause/resume).

[This has not been implemented:] As all the variables in the program are of certain data type. There is a upper limit on all the datum. If the data is too large, the result may be overflow and wrong, the reason why I do not implement this is because I can't also come up with an idea on this. Also, if the user really want to reach such a large number, the program will run for very long time. For normal users, there shouldn't be such concern.

Go to analysis

2.2 improvement for efficiency

In terms of efficiency, if the user want to test a sequence of large numbers of tosses, he may wait for a long time for every number and restart every time. I added a pause/resume function so that after each test, the user is not forced to start again from 0. For the example raised above in part 1, if user want to test 1 billion 2 billion and 3 billion, the program could only run 3 billion times after this improvement.

Go to analysis

2.3 improvement for function

In order to do achieve mass data analysis, my prototype will remember related data on every trial and show a cumulative fair rate of the generation. The toolTipKits of that textfield(fair rate) is the number of trials and fair trials. So the user could do more experiments and get a more reliable result.

To give the user more freedom in case they need, I allow the user to stop the generation, change the threshold value(this change is optional) and restart the generation from trial 0 without initialize the fair rate to 0.

If the user only want to see the deviation, I allow the applet to run even if the threshold has not been input. In this case, the threshold will be considered 0, that is, no error tolerance on the random generation. Also, I add a clear-all button to initialize everything in case the user want to restart.

[This is not been implemented]Actually, an alternative way is to let user input the number of trials and the applet will give back the fair rate. This alternative will be more direct, effective and efficient. I think this way has already changed the structure of this applet and it won't be considered as an improvement. So I do not implement it in this way.

[This is not been implemented:] As being mentioned in part 1, the random generation of other data types were not covered. For the Boolean generation part, it is quite simple because it only have two values. For other data types, for example, int data type, as there are so many aspects to be considered, what does the fairness mean is ambiguous. When referring to many mathematicians, they have provided many ways to do this, for example, there is a Discrete Fourier Transformation targeting at testing the randomness of an integer. Obviously this is too complex for this applet. So I give up to improve this applet in this aspect or approach.

Go to analysis

2.4 improvement for learnability

Add an textarea to specify function and principle for this applet as an instruction. Also, add label and set toopTipKits for every textfield to specify what it stands for. You can hover the cursor over the textfield to see the tooltipkits. As specified above, the tooltipkits for the "fair rate" will give the statistical data when the program has been running at least once.

Go to analysis

2.5 improvement for accessibility

Although a .jnlp file may cause problem, this kind of file is very useful in preparing the java environment for the applet. If there is no java environment on the device, a jnlp file will automatically install and run. In my prototype, I choose to make it a runnable jar. As this is an assignment submission, it not so likely that the user, my teacher, don't have java environment on his computer. So a .jar file is already enough.

Go to analysis

2.6 improvement for user experience

Rearrange the layout of the applet and try to make the window like a square. Arrange widgets with similar function or tends to be used together within less distance.

After implemented the functional improvements as specified above, I find there are too many buttons on it. Although some buttons will be disabled when the user is not expected to click it, too many button keep varying will still be annoying. So I combined the start/stop together on one button and pause/resume together on one button as they won't work at the same time. The interface becomes more clear and simple without reducing functions.

Go to analysis

3. New Design Model (Refer to UML.png for original image)

Please scroll down to next page for image