CS124 Individual Project - Down the Scurvy Dog Write Up

Owen Garland - owg1@aber.ac.uk

May 2, 2014

0.1 How to run this software

Download the .jar file, then either double click on it in Windows, or from the command line on any OS run. use the -text flag to use the command line version.

```
java -jar PirateApp.jar
java -jar PirateApp.jar -text
```

0.2 Analysis

The problem that I was given was to create a simple hangman like game that involved a pirate walking a plank, rather than the traditional hangman. The specification for the project recommended that I use the Java to create a text

based version of the game and then use the Swing library to create a GUI. I was also given an interface for the game model.

I was also given a small dictionary of words that would have to be used in the game. These words would need to be read in from the filesystem and then be used as the hidden words for the user to attempt to guess. This made it clear that I needed to create a class that could handle loading in text files in the given format.

After examining the interface it became clear that I would need to create a game model class that would implement the interface I was given. I would then create two views of the game model, one text based and one graphical.

0.2.1 Requirements

The application must:

- Implement the GameModel Interface provided
- Read from the piratewords.txt file provided
- Provide a graphical and text based interface
- Be pirate themed
- Have a JUnit tests written for it
- Be packaged in a JAR file

0.2.2 Runtime Environment

The application will be written in the latest version of Java, and using the Swing library. There was no target OS defined so it is assumed that it should be able to run on multiple platforms. I will be developing it on Linux, and then testing on Windows and Linux. Sadly I dont have a Apple device to test on, however a JAR file that runs on Windows and Linux should work well on Mac as well.

0.2.3 My Solution

I will be using the Model View Controller design pattern, as this will allow me to break the problem down into more logical constituent parts. The most import part of this design pattern is the model, as that is what actually performs the functions of the program. Thanks to the interface that I have been given the model will actually be quite easy to create as the needed functions and parameters are already defined for me, I just need to implement them.

The view and controller functions of this design are merged into one for this implementation as the two views that I have to create, the text and GUI versions, can easily control how the game interacts with the model; making it unnecessary to create a controlling class. This saves time and again simplifies the design.

In summary I will have the GameModel which is programmed from the interface given, and two view/controllers, one for the GUI and one for the text version of the game.

0.2.4 Use Case Diagram

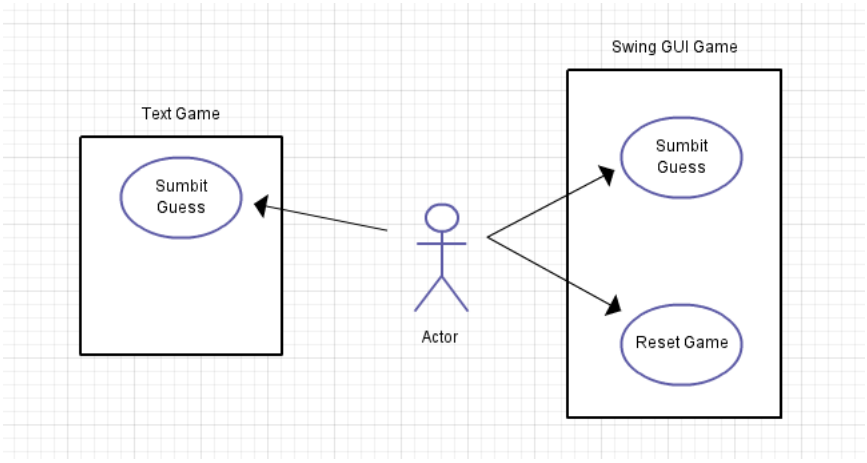

Figure 1: This shows how the user will interact with the system

0.3 Design

I have already identified the need for a few classes in my analysis of the problem. Now I will go into more detail about what the classes will contain and how they will be organised.

I have decided to only have five classes in my solution, one for running the main method and launching the application, one class for loading the pirate-words.txt dictionary file, a game model class that will implement the interface that I was given, and then a text and gui class for the two different views of the games.

0.3.1 PirateApp

This is the class that contains the Main method and runs the application. It handles a few small administration tasks such as reading in the command line arguments and creating an object of the Dictionary class for the game model to use. Then, depending on the command line options it will pass the model object across to either the SwingGame class or the Text Game class.

0.3.2 Dictionary

This class will simply load a given text file from the filesystem, and parse it. It will also provide a method for randomly selecting a word from the loaded text file. In this case it will be set to load the piratewords.txt file that has been provided; however it was also used to load a text file with just one word in for the JUnit tests.

0.3.3 TextGame

This class is actually a very small class, as it only really has one function, to take the users input from the command line, pass it to the GameModel and then display the results. It doesnt even have any methods, I will just run a While loop in the constructor that takes the input until either the game is won or lost.

0.3.4 GameModel

This is the most important class in the design as it performs all of the algorithms that the game uses. It implements the GameModelInterface that we were given in the specification. As well as the methods included in the interface I have also added a couple of methods that I thought would be useful.

Firstly I have added a won() method, when called it will return true or false depending on if the user has won the game. This makes it easier for the view classes to know when the game has been finished.

One extra feature that I decided to add was the ability to reset the game in the GUI version so the player could play again once they have won or lost the game (Or they give up on the word!). This required a new word to be selected from the Dictionary, so the selectNewWord() class will get a new random word.

0.3.5 SwingGame

This class is rather large in terms of code, that is because it will contain all of the data for how the GUI is to be presented and laid out. All of this code is contained within the createGui() method, this is just to keep it neat. The other methods are for the creation of popups or actions to be performed when a button is clicked. I have also created an AddButton() method that uses Callable to make the creation of a button and event listener/handler much easier.

0.4 Algorithm Design

I believe that the easiest way of creating the actual hangman operation of the game, a hidden word that gets revealed, will be to have two arraylists, one character array that contains the letters in the hidden word, and then a boolean array that determines whether or not the character in the word has been found.

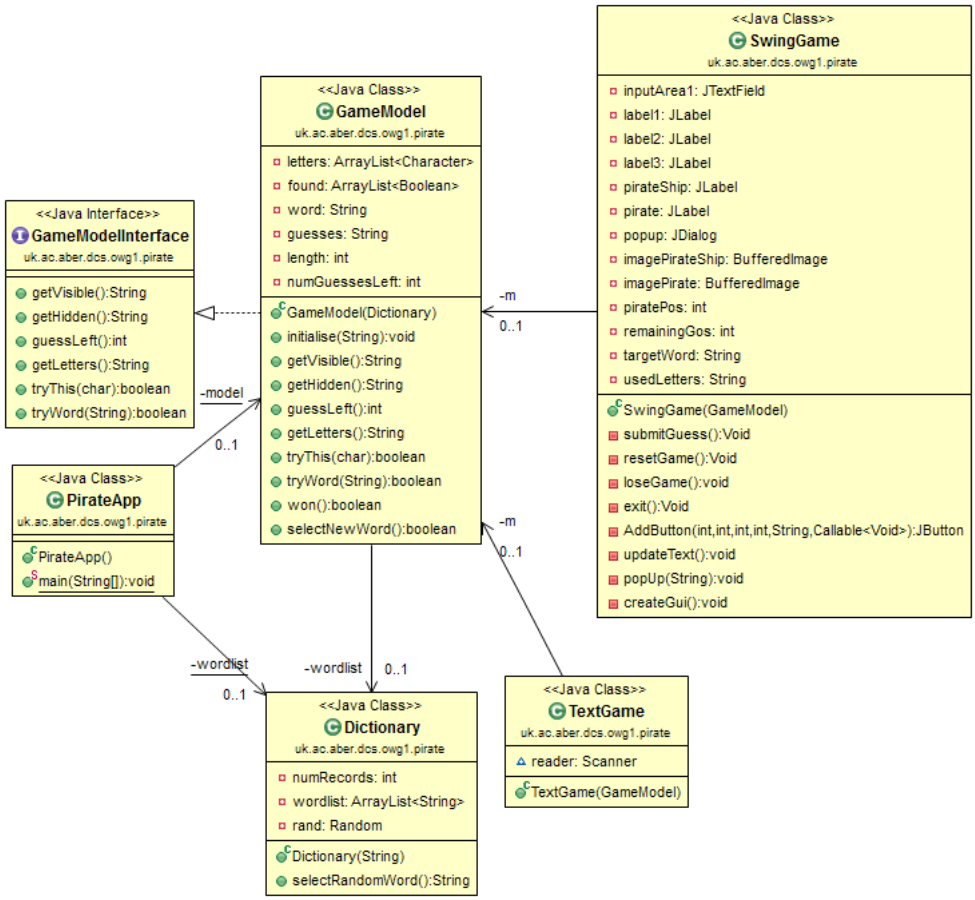

Figure 2: This shows how the different classes will be laid out

So for example, I would have two arrays, that would be worked on to produce the visible data for the user.

Boolean	Found	F	F	T	F
Char	Letters	A	H	O	Y
String	Visible	*	*	O	*

Figure 3: This demonstrates how I will hide the word

The main algorithms for this application are all contained within the GameModel class, this is programmed from the GameModelInterafce, so it is clear what algorithms will be needed. Below is some pseudo code that explains how the main algorithms will work.

0.4.1 Initialise

This is a method in GameModel that will set up the needed ArrayLists for the game, based on the word that was passed to it.

```
for each character in word
    letters.add(character)
    if(character == ' ')
        found.add(true)
    else
        found.add(false)

length = letters.size
```

0.4.2 Get Visible

This method will perform the operation that is demonstrated in figure 3. It simply performs an TRUE operation on the letters arraylist using the found arraylist.

```
for length
    if letter = ' '
        result += " "
    else if(found == true)
        result += letter
    else
        result += "*"

return result
```

0.4.3 Try This

This method takes a char as an argument called letter, and then tests whether the char is used in the word, and then if it is correct it sets the position that it is at in teh found array list to true, else it removes a guess from the number of guesses the player has.

```
correct = false;

for length
    ifletters.get(i) == letter
    found.set(i, true);
    correct = true;

if NOT correct AND letter not in guesses
    numGuessesLeft--
    guesses += letter + " "
```

```
else
    correct = true;

return correct;
```

0.4.4 Try Word

This method takes a String as an argument called guess. It simply test whether the guess is equal to the hidden word. If it is then it fill the found arraylist to true, if it isnt correct however it removes five guesses from the remaining guesses left.

```
if guess == word
    found.fill = true
    return true
else
    numGuessesLeft += -5
    return false
```

0.4.5 Won

This method isn't specified in the interface for GameModel but I think that it will make it easier to determine when the game has been won, rather than trying to calculate it in the view/controller areas. It just checks whether all of the elements in the found arraylist are true or not. If they are then the game has been won, if not then the game has not been won yet. This method returns a boolean value, true for won, false for not won.

```
for each value in found
if value != true
    return false

return true
```