CS124 Individual Project - Pirate Hangman - Write Up

Robert McHugh (rom31@aber.ac.uk)

May 4, 2014

Contents

1 Analysis

1.1 Problem and Solution Spaces

The problem space for this year's assignment was defined in the original specification found on the University's Blackboard Online Learning Platform. The main problem space from the original specification is below:

> *This project is a variation of hangman. It involves programming both a text based, and a graphics based version of the same game. The graphics based version should be pirate themed. The words to be guessed must be pirate themed and must be read from the provided file called piratewords.txt.*

I have defined the solution space to cover the whole problem space, as well as include some of the suggested functionality. This is because I believe that I should be able to provide an accurate and reliable implementation of the application defined in the original specification and that providing additional functionality will not hinder the application's performance.

1.2 Application Requirements

After reading the original specification, I have decided will need to meet the following functional requirements:

1. Create a Game Model which will hold the code that will allow the user interfaces to:

   (a) Load in the PirateWords.txt file.
   (b) Select a random hidden word from the list of words.
   (c) Encrypt the hidden word so the user cannot see the contained letters.

2. Present a text based user interface of the Hangman Game which:

   (a) Changes the guesses left as wrong guesses are made.
   (b) Presents the user with a menu that allows the user to either guess a letter or a word.
   (c) As correct guesses are made, reveal guesses letters in the encrypted word.
   (d) Shows the encrypted word for the user to guess.
   (e) Allow the user to quit the game when they please.
   (f) Allows the user to see which letters they have already guesses.
   (g) Reveals the word and tells the user when they win.

3. Present a Graphical based user interface of the Hangman Game which:

   (a) Shows the user what letters they have already guessed.
   (b) The user can enter a letter or word into a text field to guess the encrypted word.
   (c) A set of pictures show a pirate walking the plank as they make incorrect guesses.
   (d) Updates the Hidden word as correct guesses are made.
   (e) Gives a Picture to the user telling them when they either win or lose.
   (f) The user can quit the graphical game by clicking close on the window.
   (g) Allows the user to reset and get a new word to guess.

4. The graphical user interface should also implement certain aesthetic and non-functional requirements such as:

   (a) The images displayed should be pirate themed to fit in with the theme of the Pirate Game.
   (b) The labels and buttons should be appropriately named.
   (c) The layout must make visual sense.

1.3 Running Environment

From the specification I can determine some factors about the environment my implementation will be run in. I will also make some assumptions about the environment based on the information that I have not been told. These are:

1. The application will be run using the latest version of the Java Runtime Environment (Currently 1.7).
2. It would be unwise to assume that the application will be run on a certain Operating System, therefore the application should be compatible with all major Operating Systems: Windows, Linux and Mac OS X.
3. The platform that runs the application will support the Java Swing library.

1.4 My Solution

My solution to the problem space is to use the Model-view-controller design pattern when making my graphical user interface, this is where the graphical interface takes the role of controller and view and I will use the GameModel Interface that I am given in the Project Brief to create a class GameModel which will be the Model.

The GameModel will implement the GameModel Interface and contain all the necessary code to run both the graphical and non-graphical user interface. This includes loading words from the input file, select random hidden word, encrypt the hidden word, and so on.
Both the graphical and text-based user interfaces will interact with the same GameModel Class as they are just a different view of the same data. I have taken this approach as I believe that keeping all the major game code in one class simplifies things, reduces the need to create multiple methods that do the same thing and also allows for easier further expansion of the application if alternative interfaces need to be added.

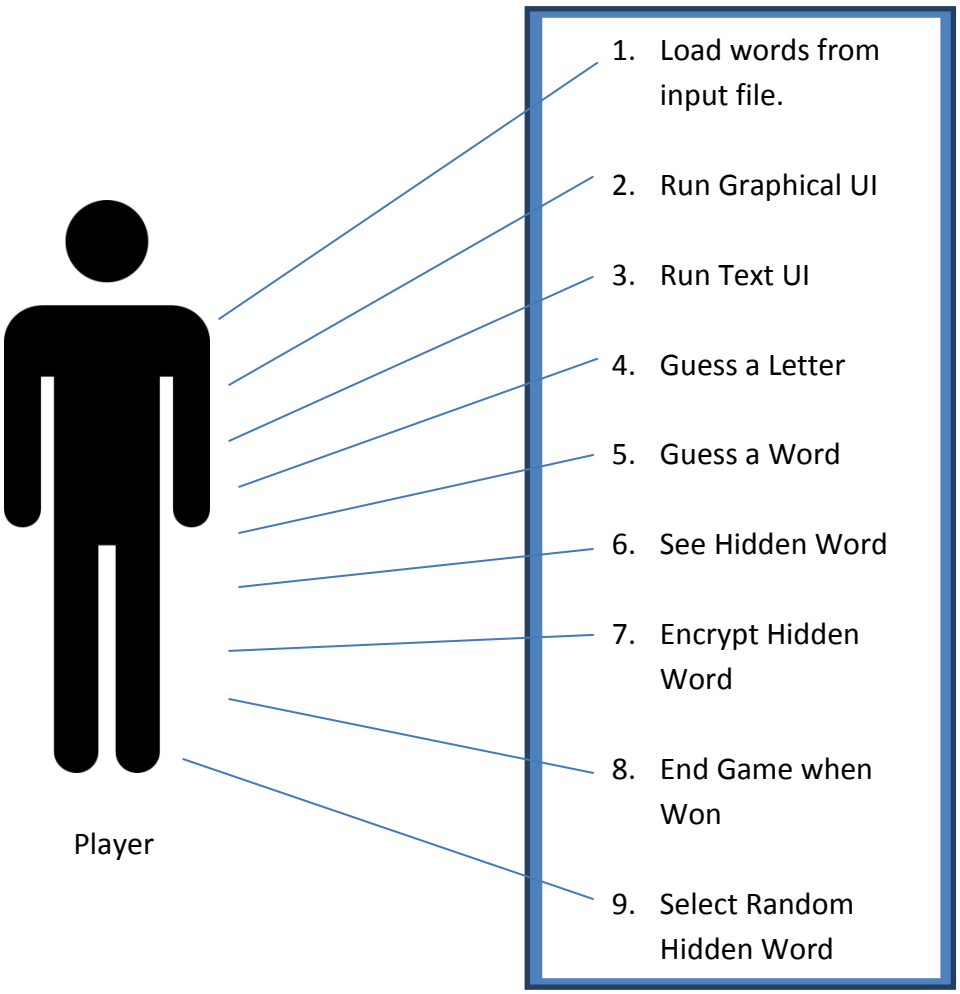


Figure 1 : Use Case Diagram

2 Design

2.1 UML Class Diagram

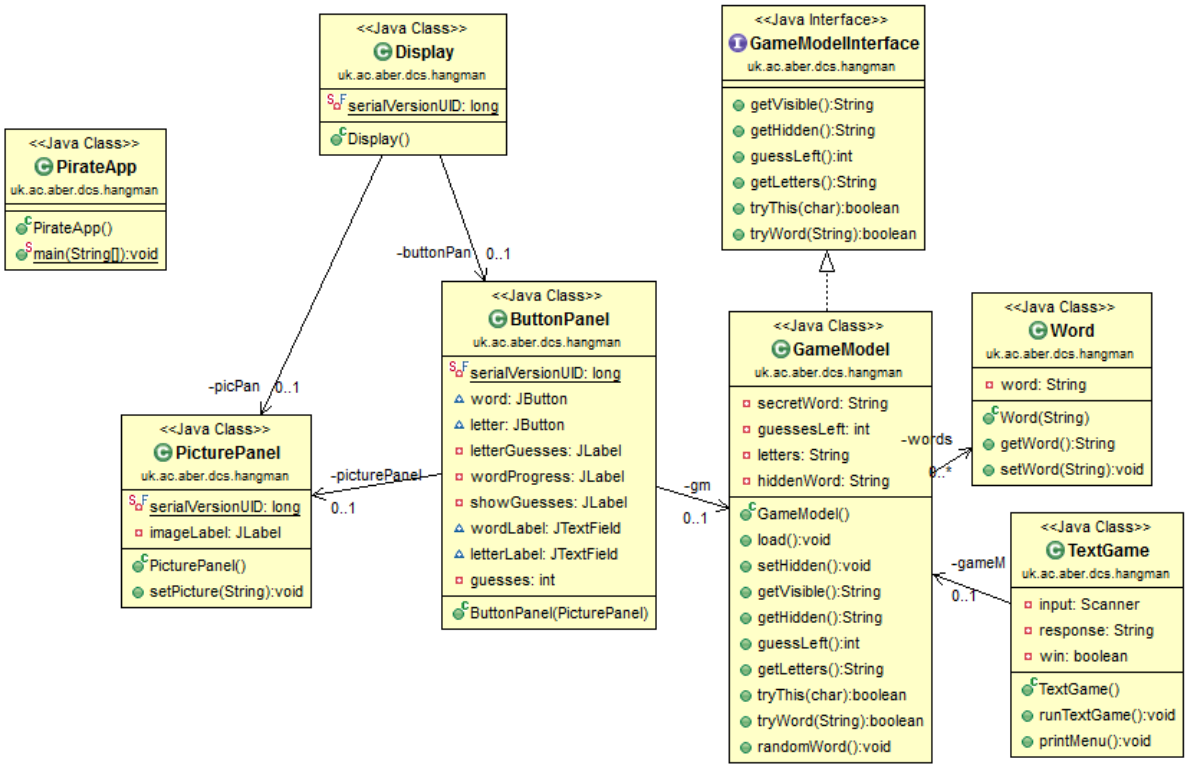


Figure 2: UML Class Diagram

The **PirateApp** class contains the main method for the application and depending on the settings put in by the user when the game is run will either create a Graphical or Text-based UI for the game.

The **GameModel** class hold all the methods to run the game such as load(), tryThis(char), tryWord(String), randomWord() as well as the ArrayList if **Words**. **GameModel** Implements **GameModelInterface** which was provided which ensures I have added all the classes needed to run the game. The classes **TextGame** and **ButtonPanel** use GameModel to run their text and Graphical user interfaces.

The class **Display** contains the code to create a **ButtonPanel** and **PicturePanel** and then set up the window settings for the JFrame that the GUI game is played in.

The class **ButtonPanel** contains the JButtons, JTextFiels and JLabels used in the Graphical User Interface. The Button also uses the class **PicturePanel** to allow the JButtons to change the image being displayed into the GUI. **ButtonPanel** also contains all the ActionListeners that allow the user to play the GUI game.

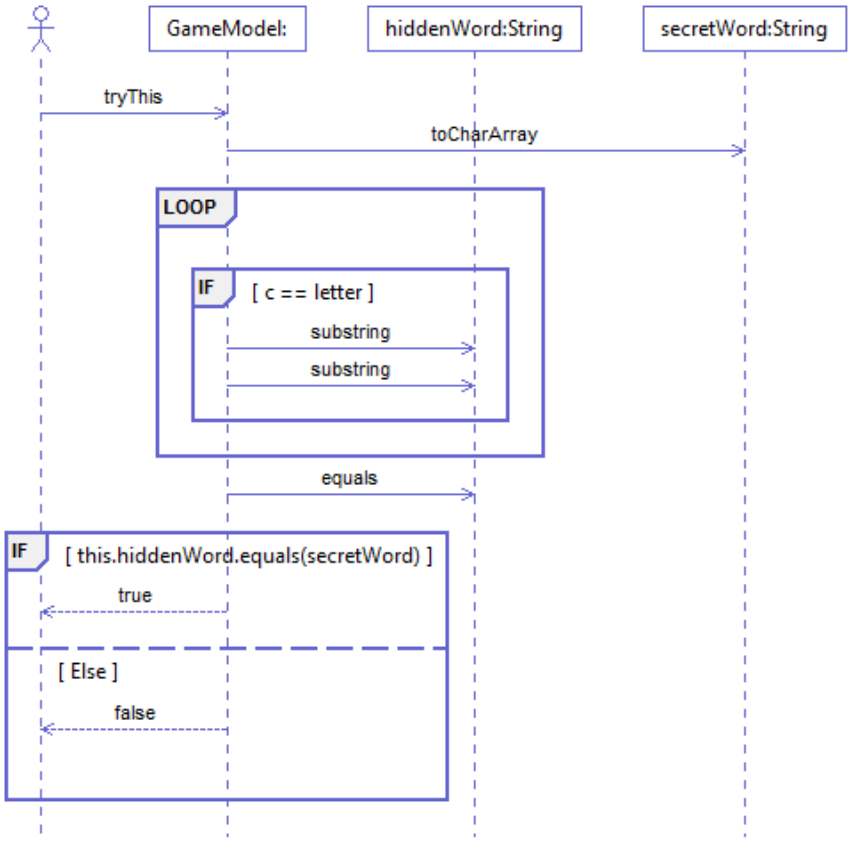
The class **PicturePanel** contains the JLabel that holds the image to be displayed in the GUI Game. **PicturePanel** also contains the method setPicture(String) that is used my **ButtonPanel** to change the image being displayed in the GUI game.

The class **TextGame** contains the methods runTextGame() and printMenu() that are used to run the Test-Based User Interface. **TextGame** also contains an input Scanner and the String response to allow the user to input what they want to do in the Text-Based User Diagram.

The class **Word** contains all the methods and variables to hold the pirate words that are loaded from the piratewords.txt file. An ArrayList of Words is contained in the class **GameModel** to hold all the loaded Words.

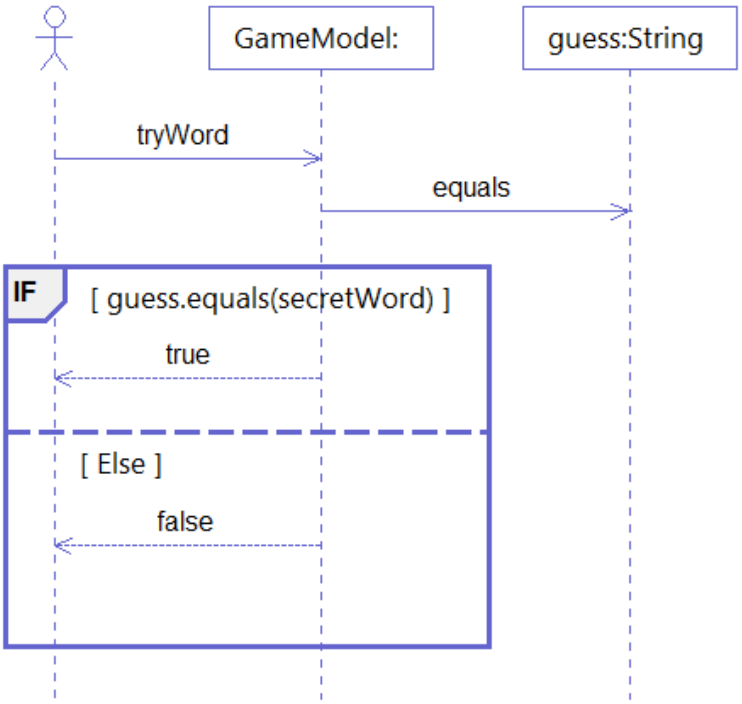
2.2 UML Sequence Diagram

tryThis() method



This is a sequence diagram showing the operations of the tryThis method in the GameModel class. The tryThis method takes in a char from the user, sets the secretWord toCharArray, the loops through and checks to see if the inputted char matches any chars in the array. Then returns true or false depending on if it finds any matches.

tryWord() method



This is a sequence diagram showing the operations of the tryWord method in the GameModel class. The tryWord method simply compares the inputted string to the secretWord String and then returns true of false depending the Strings match.

2.3 Algorithm Design

Here I will outline a few of the larger algorithms for my application. I will use pseudo-code and so will not be syntactically correct, this is more of an outline of what the algorithm will do.

2.3.1 Loading words from Input File (Figure 1: Use Case. 1)

```
Check if file exists, If fileName doesn't exist thow IOException.
 Otherwise, inFile nextInt, nextLine,
Int num =inFile Int,
for( int i=0;i<num;i++)
        String w = nextLine
        Word temp=new Word(w)
        add temp to the words ArrayList
when loop finishes, close file.
```

This algorithm will read in how many words are in the text file, and then loop through creating a temp word and then adding it to the words ArrayList. When the loop ends the text file is closed.

2.3.2 Guess a Letter (Figure 1: Use Case. 4)

```
Char letter = guessed letter;
For(char c: secretWord.toCharArray()
        If(c==letter)
                Replace char at this position with letter
        Return true
        If no match found
                Add letter to used letter String
                guessesLeft-=1
                return false
```

This algorithm takes a letter inputted by the user and then checks the secretWord to see if it contains the gussed letter. If the secretword does contain the guessed letter then the position is noted and the char at that point is replaced by the guessed letter and the Boolean true is returned. If the secretWord doesn't contain the guessed letter then the guesses is reduced by 1, the letter is added to the guessed letter String and the Boolean false is returned.

2.3.3 Guess a Word (Figure 1: Use Case. 5)

```
If(guessedWord.equals(secretWord)
        Return true
Else
        guessesLeft-=5
        return false
```

This algorithm takes a guessed word from the user and then compares it to the secretWord. If they match then return true, else reduce guesses by 5 and return false.

2.3.4 Select Random Word (Figure 1: Use Case. 9)

```
Random randomGenerator = new Random()
Int randomInt = randomGenerator.nextInt(wordsArray.size())
Word temp=wordArray.get(randomInt)
secretWord=temp.getWord()
```

This algorithm will create a new random number generator and create a number between 0 and the size of the wordsArraylist. Then the random number is used to get a Word out of the wordsArraylist which is set to a temporary Word Object. Then the secretWord String is set to the temp Word.toString method.

3. Testing

3.1 Text User Interface Tests

ID	Requirement	Description	Input	Expected Outcome	Pass/ Fail	See Figure
1	2a	Changes the guesses left as wrong guesses are made.	Wrong guess made.	Guesses left is reduced.	Pass	1,3
2			Correct guess made.	Guesses left stays the same.	Pass	1,2
3	2c	As correct guesses are made, reveal guesses letters in the encrypted word.	Correct letter guess made.	Letters are revealed.	Pass	1,2
4			Incorrect letter guess made.	No letters are revealed.	Pass	1,3
5	2e	Allow the user to quit the game when they please.	Selecting quit in the menu.	The Game is closed.	Pass	4
6	2g	Reveals the word and tells the user when they win.	Win the game.	The hidden word is revealed and the user is told that they win.	Pass	5

3.2 Graphical User Interface Tests

ID	Requirement	Description	Input	Expected Outcome	Pass/ Fail	See Figure
7	3b	The user can enter a letter or word into a text field to guess the encrypted word.	Correct letter.	Letter(s) are revealed	Pass	6,7
8			Incorrect letter	No letters are revealed. -1 guess.	Pass	6,8
9			Correct Word	Game Win.	Pass	6,10
10			Incorrect Word	No letters revealed. -5 guesses.	Pass	6,9
11	3c	A set of pictures show a pirate walking the plank as they make incorrect guesses.	Correct guess	Picture does not change.	Pass	6,7
12			Incorrect guess	Picture changes.	Pass	6,8
13	3e	Gives a Picture to the user telling them when they either win or lose.	Game Win	Win Picture is shown.	Pass	6,10
14			Game Lose	Lose Picture is shown.	Pass	6,11
15	3f	The user can quit the graphical game by clicking close on the window.	Click the close button on the window.	The GUI Game closes.	Pass	12
16	3g	Allows the user to reset and get a new word to guess.	Click the new word button	The game is reset.	Pass	13,14

4. Evaluation

4.1 Self Evaluation

For this assignment I would give myself a mark of 70/100, which would translate to an A grade.

I think this mark is a reasonable representation of the effort I have put into the project assignment. The total hours I have spent on the project are around 45 hours, which includes the original Design, Implementation and testing.

I am very happy with the end outcome of the application and I feel I have created an application that meets all the requirements of the original specification. I believe I have made good use of objects and design patterns such as the Model-View-Controller which the use of an underlying Game Model. I am happy with the way that the GUI came out. I am pleased with the layout and design. Although if I had more time I would of liked to of added more features such as allowing the user to add new words or save the game to return to later.

I am also pleased with how he text-based user interface came out, although I would have liked to add some Ascii images to improve the aesthetics of the user interface and add more functions such as an add word feature.

I feel I could have improved some things especially in the text-based user interface, but overall I am happy with the finished application.

During my time working on this assignment, I have learned so much and gained so much experience in developing Java applications using the Swing, especially with the use of ActionListeners and using the Model-View-Controller type design pattern. I also plan to do more with Java Swing Applications and expand my knowledge further.

5. Testing Screenshots

```
Load Successful
You have 10 turns left.

So far you have used the following letters:


The word to guess is: **


What would you like to do?
L - Guess a Letter
W - Guess a Word
Q - Quit the Game
```

Figure 1: Testing Screenshot

```
Enter letter to guess:
y
You have 10 turns left.

So far you have used the following letters:


The word to guess is: y*


What would you like to do?
L - Guess a Letter
W - Guess a Word
Q - Quit the Game
```

Figure 2: Testing Screenshot

```
Enter letter to guess:
q
You have 9 turns left.

So far you have used the following letters:
q


The word to guess is: **


What would you like to do?
L - Guess a Letter
W - Guess a Word
Q - Quit the Game
```

Figure 3: Testing Screenshot

```
The word to guess is: **** ******


What would you like to do?
L - Guess a Letter
W - Guess a Word
Q - Quit the Game
q
Too Hard for You?!
```

Figure 4: Testing Screenshot

```
The word to guess is: **


What would you like to do?
L - Guess a Letter
W - Guess a Word
Q - Quit the Game
w
Enter word to guess:
ye
Yes! The word is ye!
Congratulations
You win!
```

Figure 5:Testing Screenshot

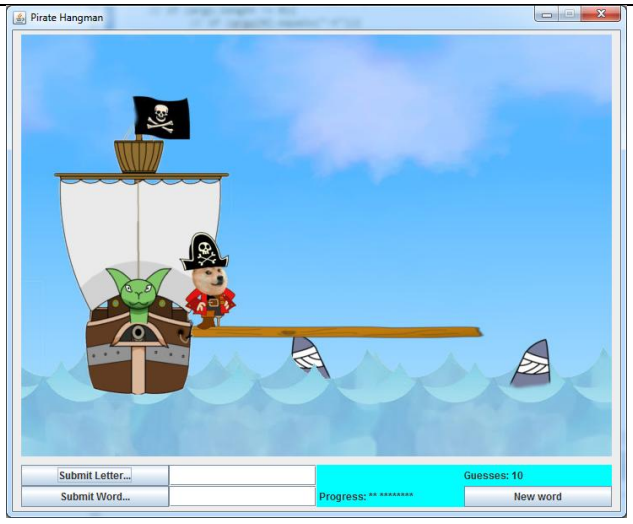


Figure 6: Test Screenshots

Figure 7: Test Screenshots

Figure 8: Test Screenshots

Figure 9: Test Screenshots

Figure 10: Test Screenshots

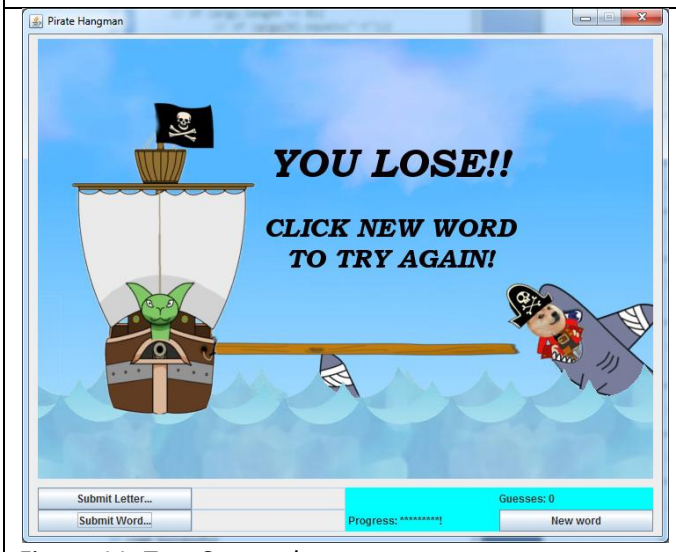


Figure 11: Test Screenshots

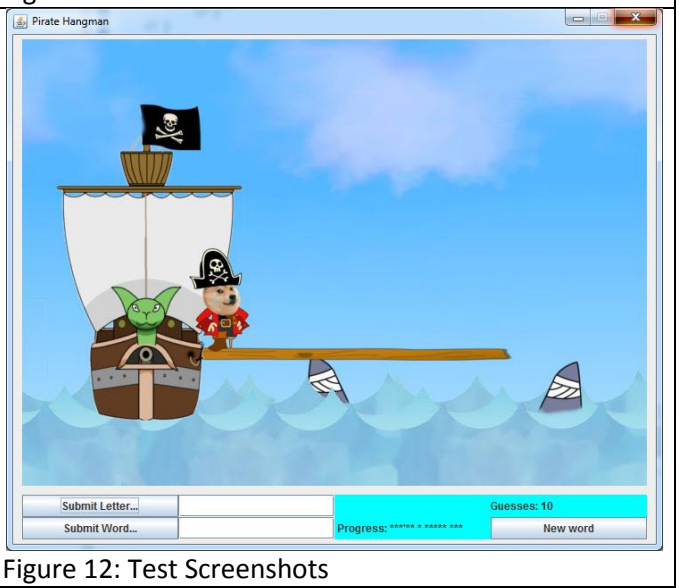


Figure 12: Test Screenshots

Figure 13:Test Screenshots

Figure 14: Test Screenshots