Tianyi Mu (tm197)
CS201
Clever Hangman Part2

By adding the debugging feature to `FrequencyGuesser`, it is clear that the guesser is functioning properly. For word length 5, the first character guessed is always the same (always "e"). Since the overall word list for Hangman is fixed, this means that `FrequencyGuesser` is recalculating the frequency of most often character for every new word it needs to guess. When the debugging function is turned on, a segment of the output looks like this:

```
word length: 5
guessing _____ using e
guessing _____ using a
guessing aa___ using s
guessing aa___ using r
guessing aar__ using o
guessing aaro_ using m
guessing aaro_ using c
guessing aaro_ using b
guessing aaro_ using n
guessing _____ using e
guessing _____ using a
guessing a_a_a using s
guessing a_a_a using r
guessing a_a_a using l
guessing a_a_a using n
guessing a_a_a using t
guessing a_a_a using i
guessing a_a_a using c
guessing a_a_a using y
guessing a_a_a using m
guessing a_a_a using k
guessing a_a_a using u
```

We can see that for the first and the second word, the order of letter guessed is different as the `FrequencyGuesser` recalculates character frequency based on character guessed. This also shows that the `FrequencyGuesser` is working properly.

By using the `BinarySearchExecutor` class, a minimal-missed table was generated for the `FrequencyGuesser`. In comparison to the `NaiveGuesser`, the `FrequencyGuesser` performs much better.

Word length	Minimal-missed
5	18
6	18
7	17
8	15
9	15
10	13

With the help of **NaiveExecutor**, a reference table for percentage of word guessed was generated. (12 misses) (Without extra credit)

Word length	Word guessed	Percentage
5	4170 words 3738 wins	89.6%
6	6166 words 5758 wins	93.4%
7	7359 words 7100 wins	96.5%
8	7070 words 7022 wins	99.3%
9	6078 words 6057 wins	99.7%
10	4591 words 4589 wins	99.96%

(10 misses) (Without extra credit)

Word length	Word guessed	Percentage
5	4170 words 2899 wins	69.5%
6	6166 words 4836 wins	78.43%
7	7359 words 6354 wins	86.3%
8	7070 words 6639 wins	93.9%
9	6078 words 5914 wins	97.3%
10	4591 words 4550 wins	99.1%

By modifying the **NaiveExecutor** and the **GuessExecutor**, a not guessed words table for eight-letter words, using 8, 9, and 10 misses until hung was generated. For extra credit, the method **removeTemplate()** and the method **match(String s, String disp)** were created in **FrequencyGuesser**.

```
private boolean match(String s, String disp)
        {
                ArrayList<String> dis=new ArrayList<String>();
                for(int i=0;i<disp.length();i++)
                {
                        dis.add(disp.substring(i, i+1));
                }
                for(int i=0;i<s.length();i++)
                {
                        if(dis.get(i).equals("_"));
                        else
                        {
                                if(!dis.get(i).equals(s.substring(i, i+1)))
                                {
                                        return false;
                                }
                        }
                }
                return true;
        }
        private void removeTemplate()
        {
                String dispWord = myGame.getDisplay();
                Iterator<String> it = words.iterator();
                while (it.hasNext())
                {
                    String s = it.next();
                    if (!match(s,dispWord)) {
                        it.remove();
                }
            }
        }
```

Here is the table for not guessed words for eight-letter words, using 8, 9, and 10 misses until hung was generated.

Guess	Missed words
8	puzzling wounding
9	
10	