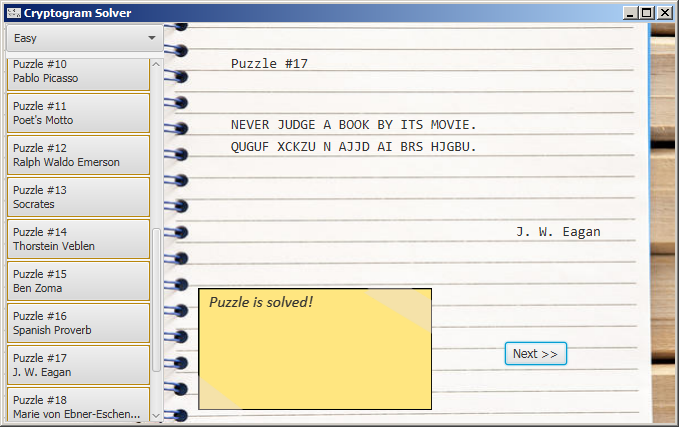
Cryptogram solver report

# Carolina kimbrough & zoey abrigo



## aBOUT

Using logic, we aim to solve cryptogram puzzles.

### What are Cryptograms?

It is a type of puzzle that consists of a short piece of encrypted text. Typically, a substitution cipher is used, where each letter is replaced by a different letter or number.

## Goal

Originally, our goal was to solve at least one hard puzzle, all easy puzzles, and a couple of medium puzzles using methods a human might use to solve a cryptogram.

However, due to time constraints and the nature of computers and natural language, we’ve made our goal a bit less ambitious: solve at least one medium puzzle and all easy puzzles.

## method

The language employed here is Java. In addition, this program utilizes a GUI interface, specifically utilizing JavaFX. The program is organized in a MVC format, model-view-controller, with **CryptoMain.fxml** as our view, **CryptoMainController.java** as our controller and the following classes in as our model:

* + **Dictionary.java** – Intended to house data and characteristics of words, contains an actual dictionary of the words collected from our quotes. This is an attempt to simulate knowledge a human being would have.
  + **PuzzleBox.java** – Contains the actual set of puzzles. Most importantly, contains the method to make puzzles out of the quotes.
  + **Quote.java** – The most busy part of the program. Originally it housed just the quote and its cryptotext equivalent but over time has developed to contain multitudes of data for each quote. It contains easy, medium and hard versions of the quote, hashmaps for character frequency, and keys that convert a character to its crypto version and back. There’s also information about the quote’s author, its length and it also has arraylists of the words in the puzzle along with its crypto equivalent and of the easy, medium and hard versions of the puzzle.
  + **Solver.java** – This part of the program does the problem-solving. An instance is generated for each puzzle and for each difficulty. This has info a human puzzle-solver might have such as a dictionary, the current quote, and various kinds of info about the quote puzzle including its difficulty and the characters found as well as a running solution in String form. There is a function for each puzzle-solving method.
  + **Word.java** – Contains information on each word

### Coming up With the puzzles

Using a .csv file of quotes from this website: <https://litemind.com/favorite-quotes/>, python was used to parse the quotes and narrow down our sample to around 40 puzzles with each quote being less than 46 characters long. Of the ~40 puzzles, only 20 are displayed in the actual program, however, all were utilized under the covers as the basis of our dictionary which was used to help the program to solve the puzzles.

### Coming up with the Cryptograms

Because we’re dealing with a substitution cipher, this was actually pretty simple to implement. In order to simplify solving, we converted all letters to uppercase, so we could just focus on the 26-letter set. First, we created an array of 26 characters for each letter of the alphabet where index 0 corresponds with ‘A’, index 1 correspond with ‘B’ all the way to index 25 corresponding with ‘Z’. We also made another array just like this, specifically it was implemented as an ArrayList. After this, we shuffled the ArrayList and that was how we came up with our key, allowing us to convert regular letters to our ‘crypto’ version of a letter.

This was done by the encrypt() method in Quote.java.

### Coming up with Easy, Medium & Hard Puzzles

When setting the character limit for each puzzle to be max 45 characters, this also included special characters and spaces as well. If we wanted to come up with different difficulty settings for each puzzle, our solution would need to be dynamic. This is because choosing a flat number of characters to reveal (10 for easy, 5 for medium, 3 for hard) would make longer puzzles more difficult and shorter ones more easy. Not to mention, the case where sentences might have a smaller set of letters (for example, ‘She sells sea shells by the sea shore’ which is long but has many of the same character).

Our solution to this was to choose a percentage of characters to reveal:

|  |  |
| --- | --- |
| Difficulty | Characters shown |
| easy | 70% |
| medium | 50% |
| hard | 20% |

But which characters do we show? The least frequent ones! So, we made a hash table of letters to the number of times they appear in the quote and sorted it by least frequent. Using the percentage above we picked the top percentage as the characters to reveal.

This was done by the findCharFreq(), toEasy(), toMedium(), toHard() methods in Quote.java.

### Solving the Actual puzzles

#### a problem

Initially, we had wanted to use the methods to solve our cryptogram that a human being would use, such as the methods on this site: <https://www.cryptograms.org/tutorial.php>

However, a problem with this is that in order to say, pick from a list of two-letter words, the way to narrow down an answer would be to understand the context of the sentence. For a computer, this is not very straightforward and could possibly involve having to tag parts of speech and a deeper understanding of the English language that would have to be ported over so that a computer could understand.

Additionally, and more fundamentally, how does a computer know what a legitimate word is? A computer understands 3,938 but it doesn’t know that ‘mptipn’ is not an actual word or that ‘motion’ is an actual word.

#### A solution

For the fundamental part, we solved that problem by giving the computer its own dictionary of words parsed from the actual set of quotes it’s setting out to solve. However, this is from the set of ~40 quotes so while, like a human, it now knows what an actual word is, its knowledge is limited to just the words in those quotes, making puzzle-solving a bit easier for it. A well-read person would have a slightly more difficult time since the breadth of their vocabulary is larger.

And, for the first problem,

The part of our program made to solve the puzzles was Solver.java. In it, there are the main methods for solving the puzzles and additional subroutines to modularize some of the repetitive bits of code in the problem-solving methods. Also, for each successful answer found, a count is made so as to report back to CryptoMainController.java for display.