**IN628 2019 Assignment 2 – Language Exploration**

**Due Date:** Friday, 22nd November, 5.00 pm – code freeze

**Value:** 30% of your final mark

**Group Size:** Individual or group of two.

**Learning Outcomes:** 1-3

For this assignment, you will use **Ruby** and **Rust** to build **two implementations** of the game **Word Mastermind**. We will be covering the basic features of **Ruby** and **Rust** formally in class; you will be learning the more complex features independently or as a group. The main purpose of the assignment is not just to build this simple game, rather it is to demonstrate your ability to effectively learn two new programming languages which differ, both programmatically and syntactically, from the familiar C-family languages.

**Word Mastermind** is a variation on the classic coloured-peg puzzle game **Mastermind**, but using words and having slightly different rules. In **Word Mastermind**, the computer **(codemaker)** chooses a word **(code)** and the player **(codebreaker)** tries to figure out the word. At each turn the **codebreaker** makes a guess. The **codemaker** provides feedback about the accuracy of the guess. Specifically, for each letter in the **codebreaker’s** guess, the **codemaker** indicates one of three outcomes:

* **Exact**: The letter is an exact match to the letter in the same position in the **code**
* **Near**: The letter is contained in the **code**, but is not in the correct position
* **Miss**: The letter is not contained in the **code**

For example:

|  |  |
| --- | --- |
| **Code** | piano |
| **Guess** | night |
| **Feedback** | near exact miss miss miss |

The **codebreaker** knows that ‘i’ is the second letter in the **code,** the first letter is not ‘n’, but is somewhere in the **code** and ‘g’, ‘h’ and ‘t’ are **not** contained in the **code**. The **codebreaker** is allowed a fixed number of guesses – the fewer the guesses, the more difficult the game. If the **codebreaker** guesses the **code** within the permitted number of guess, s/he wins the round.

In your implementation of **Word Mastermind,** you will use five letter words only. A list of words is provided as a text file, which your game must load when it is launched. You must use this; it may not be modified. The codes may only be words that contain **no duplicate letters** (e.g. piano is valid but aaron is not because it contains multiple occurrences of ‘a’). You must ensure, programmatically, that only legal words are selected from the loaded word list.

Your two versions must implement the core game play, with the specific functional requirements shown below. The code must be elegant, technically correct, architecturally sound and written in idiomatic **Ruby** and **Rust**. In addition, to demonstrate your mastery of the language syntax and semantics, you will provide detailed code commenting to explain the logic of your implementations, and to describe each of the syntactic elements you used to implement that logic.

**Functional requirements. The application must:**

|  |  |
| --- | --- |
| **1** | Be written Ruby version 2.3.1 and Rust version 1.36.0. |
| **2** | Launch without modification. |
| **3** | Be entirely console-based. Do not submit any GUI code. |
| **4** | Load its list of potential words from an external text file (word-list.txt) provided when it is first launched. The word list may not be modified. |
| **5** | Randomly select a word **(code)** at each round. This word must not contain duplicate letters. |
| **6** | Allow a fixed number of guesses for each round. Each guess is a five-letter word from the keyboard by the player. |
| **7** | Provide feedback about the letters in the guess as described above. You may use whatever text-based display format you like. Make sure it is clear to the player. |
| **8** | After each guess, display the number of remaining guesses in some way. |
| **9** | Clearly indicate a win or loss. |
| **10** | Allow the user to play as many rounds of **Word Mastermind** as s/he wishes, exiting with a specific keystroke. |
| **11** | Fulfil the special commenting requirements discussed below. |

**Marking Schedule:**

Attached at the end of this document.

**Group Contribution:**

All git commit messages must identify which member (or members) of the two participated in the associated work session. Proportional contribution will be determined by inspection of the commit logs. If the commit logs show evidence of significantly uneven contribution proportion, the lecturer may choose to adjust the mark of the lesser contributor downward by an amount derived from the individual contributions.

**Submission:**

* Project files must be submitted via GitHub Classroom. Here is the link to the repository you will be using for submission – <https://classroom.github.com/a/5ZRJ-y9I>
* Your primary code files must be named ***<your\_op\_username>.rb***and ***<your\_op\_username>.rs*.** Only files with this exact naming format will be marked.
* If you submit a multi-file solution, you are responsible for ensuring that all secondary files are correctly included in the build – you will need to explore the **Ruby** and **Rust** syntax for this.
* A minimum of two commits per week is required. Insufficient commit frequency is grounds for rejection of the submission and award of zero marks for the assignment.

**Commenting:**

A stated above, the primary purpose of the assignment is to demonstrate your ability to learn and use two new programming languages. The most direct way for you to demonstrate your mastery of **Ruby** and **Rust** is to explain your code thoroughly via comments. In this assignment, your code comments are not for future reference, or for the convenience of the reader, as per normal. Your code comments are where you demonstrate how well you understand the code you are submitting. To gain the full marks for commenting you must have:

* A header comment for each method, which explains in detail the input, output, effect and computational logic of that method.
* Inline commenting for every computational statement which explains in detail the syntax and logic of the construct.
* Inline commenting every **Ruby** and **Rust** syntactic structure, which explains in detail each element of the construct and its role or function.

Make sure your comments don’t simply translate the **Ruby** and **Rust** commands into English. You must explain both **what you are doing** and **why you are doing it**. A fully commented submission will be completely clear, at both the syntactic and semantic levels, to a reader who has never seen Ruby or Rust code before

**IN628 2019 Assignment 2 – Language Exploration Marking Schedule**

# Author(s):

# Mark:

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Weight** | **Earned** | **Total** |
| Code commenting | 35% | /15 | 0 |
| Program Structure | 20% | /15 | 0 |
| Code Quality | 20% | /20 | 0 |
| Functionality & Robustness | 25% | /40 | 0 |
| **Total** |  | **/90** | **/100** |

**Award metric: Very Poor = 0; Poor = .25; Ok = .5; Good = .75; Very Good = 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Code Commenting** |  | **Max** |  |
| Header comment for each method |  | 5 |  |
| Inline comment for every computational statement |  | 5 |  |
| Inline comment for every syntactic structure |  | 5 |  |
| **Program Structure** |  | **Max** |  |
| General class architecture |  | 5 |  |
| Sufficient modularity |  | 5 |  |
| Efficient algorithmic logic |  | 5 |  |
| **Code Quality** |  | **Max** |  |
| Idiomatic Ruby and Rust |  | 10 |  |
| Correct flow of control |  | 5 |  |
| Correct data structures |  | 5 |  |
| **Functionality & Robustness (penalties assessed for bugs, omission or failure to meet spec)** |  | **Max** |  |
| Written in Ruby and Rust using version 2.3.1 and 1.36.0 |  | 1 |  |
| Opens and runs without modification |  | 2 |  |
| Words loaded from an external text file |  | 4 |  |
| Random word selected at each round |  | 3 |  |
| Word must not contain duplicate letters |  | 4 |  |
| Fixed number of guesses per round |  | 2 |  |
| Correct number of letters per guess |  | 4 |  |
| Guess feedback computed |  | 2 |  |
| Guess feedback displayed clearly |  | 2 |  |
| Remaining guesses computed |  | 2 |  |
| Remaining guesses displayed clearly |  | 2 |  |
| Win/loss computed |  | 2 |  |
| Win/loss displayed clearly |  | 2 |  |
| Play another round with a keystroke |  | 2 |  |
| Exit with a keystroke |  | 2 |  |
| No exceptions thrown or other crashes |  | 4 |  |