# IN628 2018 Assignment 2: Language Exploration

# Ruby Word Mastermind

Due Date: Friday, 16th November, 5.00 pm

Value: 25% of course mark

Group Size: This is an individual assignment. You must submit your own work, and **clearly cite** any work of others.

For this assignment, you will use Ruby to implement the game Word Mastermind (rules described below). We will not be covering Ruby formally in class; you will be learning it independently. The main purpose of the assignment is not just to build this simple game, rather it is to demonstrate your ability to effectively learn a new programming language which differs, both paradigmatically and syntactically, from the familiar C-family.

Word Mastermind is a variation on the classic Mastermind coloured-peg puzzle game[[1]](#footnote-1), but using words, and having slightly different rules. In Word Mastermind, one player (the "codemaker"; the computer in your implementation) chooses a word (the "code") and the other player (the "codebreaker"; the user in your implementation) tries to figure out the word. At each turn, the codebreaker makes a guess. The codemaker then provides feedback about the accuracy of the guess. Specifically, for each letter in the codebreaker's guess, the codemaker indicates one of three outcomes[[2]](#footnote-2):

* 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:

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

Now the codebreaker knows that 'i' is the second letter of the code, the first letter is not 'n' but there is an 'n' somewhere in the code, and the code contains neither g, h, nor t. The codebreaker uses the feedback to refine his or her next guess, circling in on the code. The codebreaker is allowed some fixed number of guesses -- the fewer guesses, the more difficult the game (around 10 guesses is typical). If the codebreaker guesses the code within the permitted number of guesses, s/he wins the round.

In your implementation of Word Mastermind, you will use only five-letter words. A list of words is provided as a text file on the I: drive, which your game must load when it is launched. **You must use this list; it may not be modified**. In our version of the game, the secret codes may only be words that **contain no duplicate letters** (e.g. *piano* is permitted but *vivid* is not, because it contains multiple occurrences of 'v' and of 'i'). You must ensure, programmatically, that only legal code words are selected from the loaded word list.

Your version 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. In addition, to demonstrate your mastery of the language syntax and semantics, you will provide especially detailed code commenting to explain the logic of your implementation, and to describe each of the syntactic elements you used to implement that logic (details below).

There is a great deal of information available to help you learn Ruby. The Robertson and University Science Libraries both have books on Ruby, and there is a wealth of material online. As usual when dealing with the internet, be careful of quality -- there's some truly execrable content out there claiming to be educational.

**Some useful sites:**

|  |  |
| --- | --- |
| The main Ruby website | https://www.ruby-lang.org/en/ |
| A convenient Ruby installer for Windows | http://rubyinstaller.org/ |
| A well-regarded Ruby style guide | https://github.com/bbatsov/ruby-style-guide |
| *Head-First Ruby*, a useful, yet amusing Ruby textbook | http://proquest.safaribooksonline.com/9781449372644 |
| *Learning Ruby*, a more traditional Ruby textbook | http://proquest.safaribooksonline.com/9780596529864 |
| A set of online Ruby tutorials | https://www.codecademy.com/learn/ruby |

**Functional Requirements. The application must:**

|  |  |
| --- | --- |
| 1 | Be written in Ruby version 2.3.1. or later |
| 2 | Be entirely console-based. Do not submit any GUI code. |
| 3 | Allow the user to play as many rounds of Word Mastermind as s/he wishes, exiting with a specified keystroke. |
| 4 | Load its list of potential words from the external text file mastermindWords.txt provided on the I: drive when it is first launched. The word list may not be modified. |
| 5 | Randomly select a code word at each round. This word must not contain duplicate letters. |
| 6 | Allow a fixed number of guesses in each round. Each "guess" is a five letter word entered from the keyboard by the player. |
| 7 | Provide feedback about the letters in the guess word as described above. You may use whatever text-based display format you like, as long as it is clear to the player. |
| 8 | After each guess, display the number of remaining guesses in some way. |
| 9 | Clearly indicate win or loss. |
| 10 | Fulfill the special commenting requirements discussed below. |

**Submission Requirements:**

* All code must be original. Do not copy anyone else's code, or use any code that you have seen elsewhere but do not completely understand.
* All work is to be submitted via your Gitlab repo as usual. Please be sure to label your assignment folder(s) carefully, so I know what you wish to have marked.
* Your primary code file (the one we will run for marking) **must** be named *yourOPUserName*\_Ruby.rb. Only files with this exact naming format will be marked.
* If you submit a multi-file solution, you are responsible for insuring that all secondary files are correctly included in the build (you will need to explore the Ruby 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 Requirements:**

As stated above, the primary purpose of this assignment is to demonstrate your ability to learn and use a new language. The most direct way for you to demonstrate your mastery of basic Ruby is to explain your code thoroughly via comments. Thus, 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 (i.e. code that does something) which explains **in detail** the **syntax** and **logic** of the construct.
* Inline commenting for every Ruby syntactic structure, which explains in detail each element of the construct and its role or function.

Make sure that your comments don't simply translate the Ruby 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 before***.* It may help you to think of these comments as a Ruby tutorial, rather than the usual programmer's tool. An example of this style of commenting can be seen at https://www.safaribooksonline.com/library/view/the-ruby-programming/9780596516178/ch01s04.html, which is an excerpt from the textbook *The Ruby Programming Language*, by Flanagan and Matsumoto[[3]](#footnote-3).

**Marking Schedule:**

|  |  |  |
| --- | --- | --- |
| **Component** | **Full Credit Criteria** | **Value** |
| **Code commenting** | * As described above | 40% |
| **Program structure** | * Modularity (including parameterisation) * Efficient algorithmic approach | 20% |
| **Code quality** | * Idiomatic Ruby * Sensible flow of control * Correct data structures | 20% |
| **Functionality & Robustness** | * All FRs implemented | 20% |

1. https://en.wikipedia.org/wiki/Mastermind\_(board\_game) [↑](#footnote-ref-1)
2. Note that these rules are different from those of colour-peg Mastermind because of the larger set of possibilities in each location (i.e. 26 letters vs. 6 colours). [↑](#footnote-ref-2)
3. The creator of Ruby [↑](#footnote-ref-3)