By: Keenan Jade Turner

README - Hangman: Escape from Alcatraz

How to run *Hangman: Escape from Alcatraz*

- *Hangman: Escape from Alcatraz* is a terminal game application created in the most recent version of the programming language, Ruby.
- Below you will find instructions for both Mac and Windows users on how to find your terminal, how to install the latest version of Ruby, and how to run the game.
- **Find your terminal / command prompt:**
  - **Mac user instructions:**
    - 1) Open ***Finder***
    - 2) Select ***Applications*** then choose ***Utilities***
    - *3) Double click on* ***Terminal***
    - 4) Terminal window should pop up
    - Visual instructions HERE
  - **Windows user instructions:**
    - 1) Open the ***Start Menu***
    - 2) Type ***"command"*** in the search field
    - 3) ***Command Prompt*** should show up under the category named "Programs"
    - 4) Click the program ***Command Prompt***, and the window should appear
    - Visual instructions for Windows XP, Vista, 7, 8 and 8.1 HERE
- **You will need Ruby's 2.2.3 version or higher** in order to run this terminal game application.
- **Checking Ruby version and installing Ruby instructions:**
  - **Mac users:**
    - **Note:** On OS X El Capitan, Yosemite, Mavericks, and macOS Sierra, Ruby 2.0 is included**.** OS X Mountain Lion, Lion, and Snow Leopard ship with Ruby 1.8.7.
    - To check the version of Ruby you are working with you will need to go to your terminal application and enter the bash command ***ruby -v***.
    - If you do not have the latest version of Ruby then you will need to use Homebrew to install the latest version.

- In your terminal, type the following command ***brew install ruby***.
  - This will install the latest version of Ruby.
  - Here is documentation on Homebrew for Ruby installation.

- **Windows users:**
  - Head to rubyinstaller.org
    - More about the RubyInstaller Project:
      - It provides a self-contained Windows-based installer that contains a Ruby language execution environment, a baseline set of required RubyGems and extensions, and the full text of The Book of Ruby.
  - Click the large red button that says “download” and it will lead you to a page with different versions to install, explanations on each version, and advice for if you have issues.
  - Download “Ruby 2.3.3”, and install it to your computer.
  - Head to the command prompt and check the Ruby version with ***ruby -v***. You should have the latest version of Ruby.

- **Steps for running *Hangman: Escape from Alcatraz***
  - Place the unzipped file named ***hangman_escape_from_alcatraz.rb*** on your desktop
  - Open your terminal / command prompt (access instructions are above)
  - Type the command ***cd ~/Desktop*** to go to your desktop while in your terminal / command prompt
  - Then type ***ruby hangman_escape_from_alcatraz.rb*** to start up the program in the terminal / command prompt
  - The game will begin with a story then ask you how lucky you are feeling, which controls the difficulty of the word.
  - After you choose the level of difficulty your game will start.

Thought Process for *Hangman: Escape from Alcatraz*

- First, I had to **understand the problem and the constraints** of the exercise then create an **algorithm for how the program should run** based on the constraints. The algorithm led to **pseudo code being crafted**, checking the flow of the program and possible roadblocks. Finally, I made **examples** to feel the behavior and see what the output could look like.
  - **Understanding the problem and the constraints**
    - On the first day of the challenge I did not code instead I reread the prompt, took notes on each section, and defined the constraints.
    - I named them “Game Mechanic Constraints”, “Game Rule Constraints”, “User Interface Constraints”, “Implementation Constraints”, and “API Constraint”.
    - This forced me to upload the problem before trying to address it with code, utilizing my conscious and unconscious mind to create solutions.
  - **Algorithm for program execution**
    - The algorithm was designed to solve the core problem.
      - Get the data from the api
      - Present word, image, guesses, and incorrect guesses
      - Take the user's choice, check to see if it is correct, invalid and or if it's more than one chars
      - Update word then display, image, guesses, and incorrect guesses view
      - Repeat this until Computer or User wins
        - Lose condition is when the user runs out of guesses, and the word still has underscores
        - Win condition is 6 or less guesses are left, and the word does not have underscores in it
      - Present a message asking if they want to play again
  - **Crafted pseudo code to flush out data structures, and flow**
    - *Examples of the pseudo code for setting the variables*

      START

      SET word to word_dictionary_api = STRING

      SET valid_choices = ARRAY

      SET win? condition = Method

SET user_guess = STRING

SET # of guess = INT

SET incorrect_guess_list = STRING

- *Tackling the UI constraints:*

  PRINT image of hangman with 6 guesses

  PRINT word

  PRINT # of guesses

  PRINT incorrect_guess_list

  PRINT ask for a choice (needs validation)

- *Handling the user's choice:*

  IF INVALID CHOICE:

    valid_choice_check invalid basically [0-9, symbols]

    PRINT invalid_choice_message

    PRINT word (state: unchanged)

    PRINT guesser_choice_list

    PRINT number_of_guesses

    PRINT hang_man_image depending on the number of guesses

  IF CORRECT CHOICE:

    valid_choice_check

    PRINT correct_choice_message

    PRINT word (state: updated)

    PRINT guesser_choice_list

    PRINT number_of_guesses

- **Examples to feel the behavior and see potential output**
  - *See what the program would look like if the user guessed incorrectly (the word is "test")*

```
 +---+-
 |   |
 |   0
 |
 |
-+----------
```

WORD: _ _ _ _

INCORRECT LETTERS: ["a"]

GUESSES: 5

Guess a letter (a-z): a

  - *Now to feel how the program would be if the guess is correct (the word is "test")*

```
+---+-
 |   |
 |   0
 |
 |
-+----------
```

WORD: _ e _ _

INCORRECT LETTERS: ["a"]

GUESSES: 5

Guess a letter (a-z): e

- *Checking the win condition (the word is “test”)*

```
 +---+-
 |   |
 |   0
 |   |
 |  /
-+----------
```

WORD: t e s t

INCORRECT LETTERS: [a, i, n]

GUESSES: 3

{[You win!]}

Play again? (press 'y' to continue)

- *Checking the lose condition (the word is “test”)*

```
+---+-
 |   |
 |   0
 |  \ | /
 |  /   \
-+----------
```

WORD: _ e s _

INCORRECT LETTERS: [a, i, n, l, z, r]

GUESSES: 0

[You lose.]

Play again? (press 'y' to continue)

- Each step of the **non-coding process** helped me turn an idea into steps to solve the problem, check for potential bugs, handle the constraints, and get a decent understanding of the scope of work.

Creative Concept & Implementation

- My goal was to have a story driven approach to a classic game.
  - Version 1.0 of the hangman game was very close to how the game would be played on a whiteboard at any local elementary school.
    - **Example:**

      +---+-

      | |

      |

      |

      |

      -+----------

      WORD: _ _ _ _

      INCORRECT LETTERS: []

      GUESSES: 6

      Guess a letter (a-z):
    - Pretty typical game layout for a hangman game. I felt this satisfied the basic requirements for the challenge.
  - Version 2.0 I wanted to have a story that engaged the user, built a quick emotional attachment to the character, and the character's success.
  - **Story breakdown**
    - I had to think about reasons for *why someone would be hanged*. The only scenario that I felt comfortable using was of a prisoner in jail.
    - **Protagonist:** We don't know what the prisoner did, but they're on a island jail, and are close to their release date.
    - **Antagonist**: The Warden is the computer. He will ask the prisoner to guess the word that he is thinking for his freedom. The player should wonder why the Warden has taken it upon himself to hang this inmate.
    - **Setting:** Bay Area, 1963 at Alcatraz island then the Golden Gate Bridge.
    - **Ending:** The game ends asking if the user wants to play again with an image showing a win or a lost.

- **Bringing the story to life**
  - With the story in mind I needed art. I found a repository of free ASCII art that had everything I needed: a person in jail, a jail on an island, a boat, and a bridge.
  - Then I had to put the images in order, create a narrative, and figure out how to sequence images so the story could progress, and feel alive.
  - I created a function called *start_story* that would store all of the print statements that held the multiline strings containing the images and text.
  - To build a sense of story progression I used the built in function *sleep#.* This function tells the program to stop for a certain amount of time (seconds), letting the player read the text, and slowly figure out what is going on.
  - Finally, to give the program a movie-like behavior I created a function called *clear_screen.* It clears the terminal of the last image, while outputting the next image. This makes it so the user does not have to scroll, but instead just sit back and watch.
- **Minor details in the game**
  - March 20th, 1963 is the day before Alcatraz closed.
  - Alcatraz is in the name of the program, but not in the actual story. Leading the player to assume, but not fully know for sure.
  - Until they get to the bridge, which resembles the Golden Gate bridge that the player fully realizes this game takes place in San Francisco.
  - The number of seagulls is equal to the number of guesses the player has left.

- **Final Thoughts:**
  - I hope you enjoy playing the game as it was a blast creating it!
  - Hangman is a classic game that children love, and it inspired me to think about how I could make it more interactive for adults to play.
  - The process of thinking through the constraints and creating a framework to create the program I felt was the most important step. Planning before executing is key!
  - Breaking down the problem into smaller problems to solve then bringing the smaller solution back to help solve the bigger problem really was key in progressing the program.