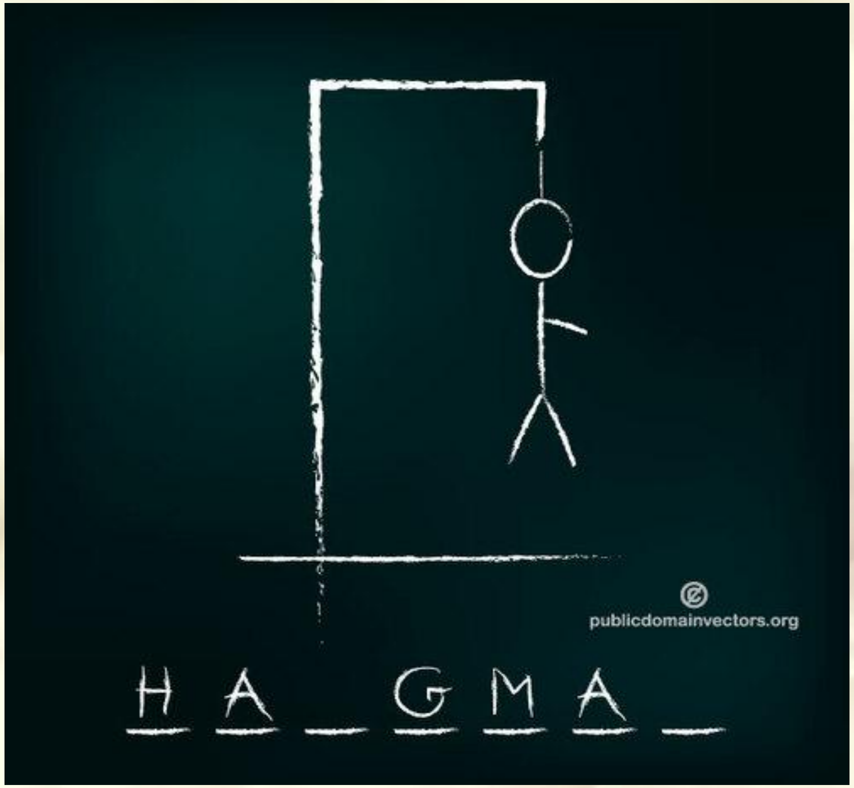
HANGMAN-GUESSING GAME



CHAITEE DUTTA GUHA

MY FIRST PROJECT IN RUBY WITH GEMS

Description of Hangman game

Hangman is an educational word guessing game to be played in a command-line interface.The game has been created with Ruby.When the Ruby script is called, it will start a new game and provide the user with a blank word and a clue. The script takes the user input (letter) and checks it against the selected word for the game. Make too many wrong choices and you will lose. Get all the letters correct to win! I use conditionals, string interpolation, classes, arrays and looping to put our Ruby skills to the test.

PROJECT PLAN

1. BRAINSTORM
2. PLAN
3. DEVELOP
4. TEST
5. PRESENT

@ BRAINSTORM

1.Bounce my ideas around and narrow down to one or two that I like.

2.Discuss with my Educator’s & Finalize.

3.Then start planning to build up my application.

@PLAN

a. Clearly define solution: What is it? What will it do?

b. Describe the flow of our application. What might the user do or expect and how will they interact with it.

c. Decide which features are part of core functionality and which should be stretch goals to be completed if time allows.

e. Create a workflow diagram or flowchart to describe the order of operations. f. Write pseudocode for the application

@ DEVELOP

WRITE CONTENT

IMPLEMENT REQUIRED FILE STRUCTURE.

WRITE CODE FOR STRETCH GOAL & core features.

Development tool is VS code.

@TEST

1. Manually test the application

2. Fix any bugs that arise

3. Try to break the application with user interactions.

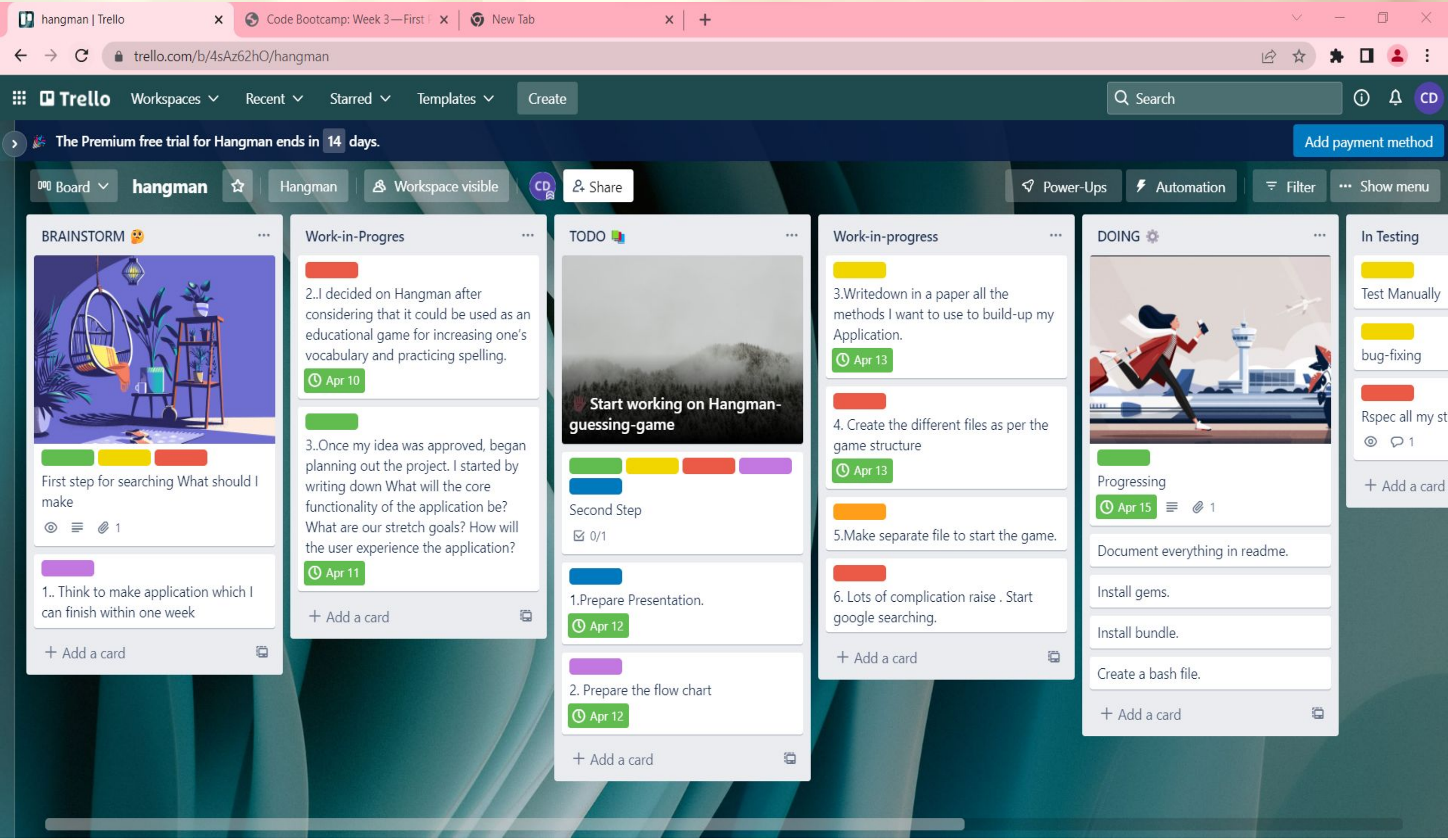
@PRESENT

Create a google doc presentation to show demo of the presentation.

TRELLO PROJECT MANAGEMENT

TIME FRAME

11-04-2022 TO 24-04-2022



BRAINSTORM

- First step for searching What should I make
- 1.. Think to make application which I can finish within one week

Work-in-Progres

- 2..I decided on Hangman after considering that it could be used as an educational game for increasing one's vocabulary and practicing spelling. Apr 10
- 3..Once my idea was approved, began planning out the project. I started by writing down What will the core functionality of the application be? What are our stretch goals? How will the user experience the application? Apr 11

TODO

- Start working on Hangman-guessing-game
- Second Step 0/1
- 1.Prepare Presentation. Apr 12
- 2. Prepare the flow chart Apr 12

Work-in-progress

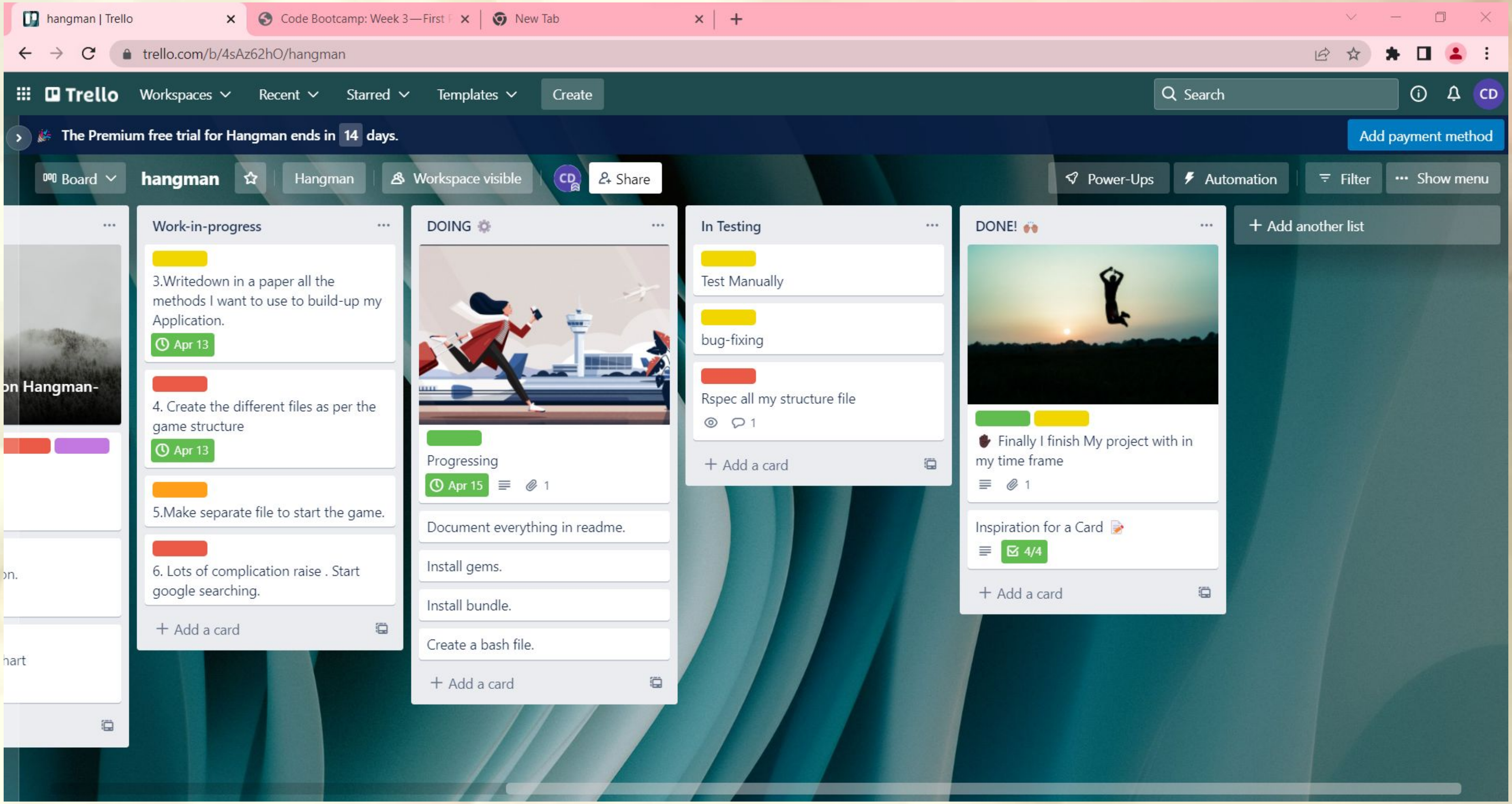
- 3.Writedown in a paper all the methods I want to use to build-up my Application. Apr 13
- 4. Create the different files as per the game structure Apr 13
- 5.Make separate file to start the game.
- 6. Lots of complication raise . Start google searching.

DOING

- Progressing Apr 15
- Document everything in readme.
- Install gems.
- Install bundle.
- Create a bash file.

In Testing

- Test Manually
- bug-fixing
- Rspec all my st



on Hangman-

on.

hart

Work-in-progress

3.Writedown in a paper all the methods I want to use to build-up my Application.
Apr 13

4. Create the different files as per the game structure
Apr 13

5.Make separate file to start the game.

6. Lots of complication raise . Start google searching.

+ Add a card

DOING

Progressing
Apr 15 1

Document everything in readme.

Install gems.

Install bundle.

Create a bash file.

+ Add a card

In Testing

Test Manually

bug-fixing

Rspec all my structure file
1

+ Add a card

DONE!

Finally I finish My project with in my time frame
1

Inspiration for a Card
4/4

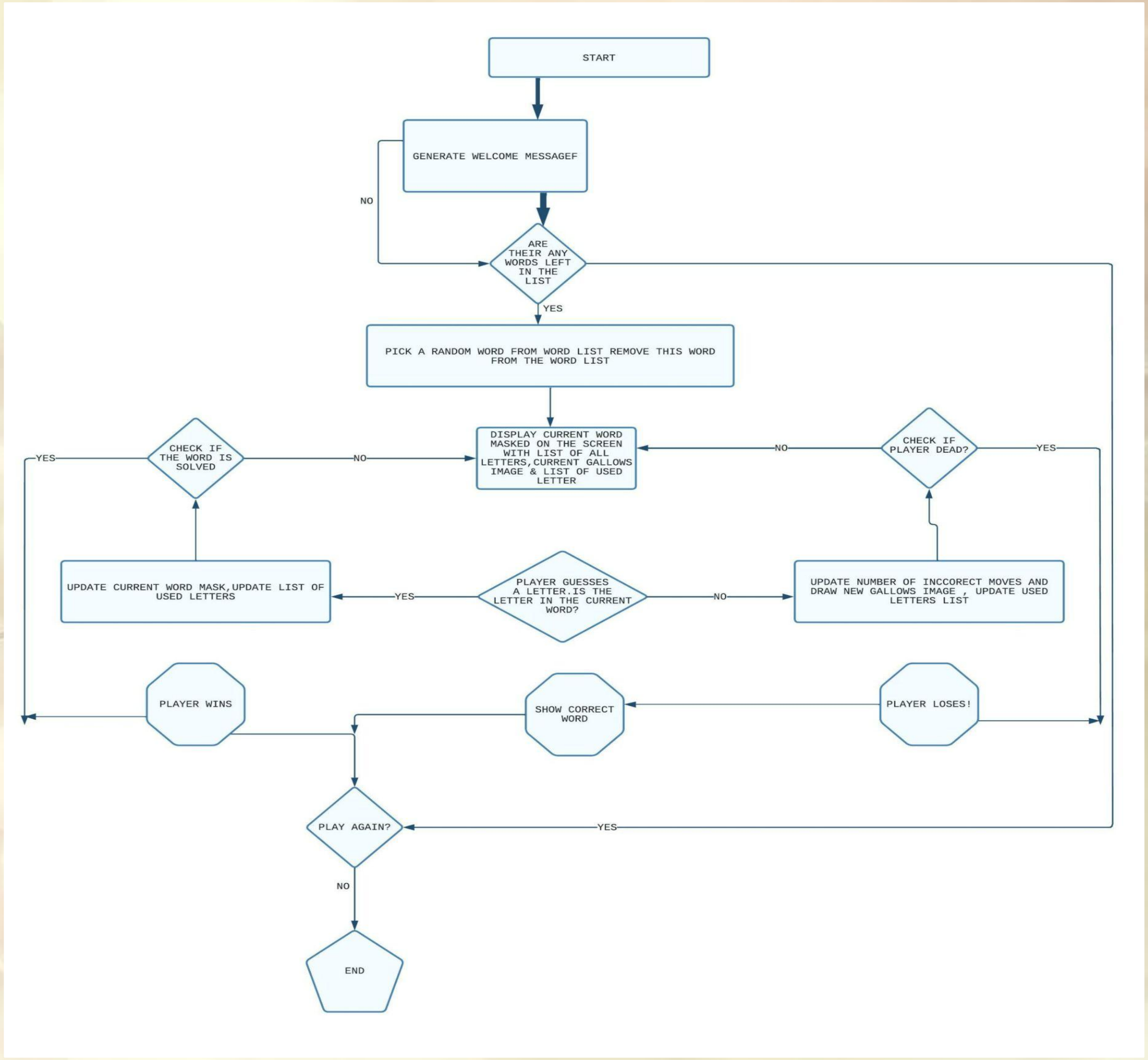
+ Add a card

+ Add another list

Functionality

- Start Program
- Program greet the user.
- Program randomly generates a question as a clue of the answer.
- PromPts the user the length of the words of that answer just as a clue to guess.
- Prompts the user for input a letter
- check user input ,if correct display letter in Answer word.if incorrect ,display number of live left once game over it will show the message "Game over... better luck next time!"
- When its win the game it will show "congratulation you have won this round.
- Again prompts about user want to play again ,Yes/No.If user press yes then back to start game , if user press no then exit from the game.

Flow Chart


START
GENERATE WELCOME MESSAGEF
NO
ARE THEIR ANY WORDS LEFT IN THE LIST
YES
PICK A RANDOM WORD FROM WORD LIST REMOVE THIS WORD FROM THE WORD LIST
DISPLAY CURRENT WORD MASKED ON THE SCREEN WITH LIST OF ALL LETTERS,CURRENT GALLOWS IMAGE & LIST OF USED LETTER
CHECK IF THE WORD IS SOLVED
YES
NO
CHECK IF PLAYER DEAD?
NO
YES
UPDATE CURRENT WORD MASK,UPDATE LIST OF USED LETTERS
YES
PLAYER GUESSES A LETTER.IS THE LETTER IN THE CURRENT WORD?
NO
UPDATE NUMBER OF INCCORECT MOVES AND DRAW NEW GALLOWS IMAGE , UPDATE USED LETTERS LIST
PLAYER WINS
SHOW CORRECT WORD
PLAYER LOSES!
PLAY AGAIN?
YES
NO
END

FEATURES

1. ARRAY:

Ruby arrays are groupings of almost any entity that are sorted and integer-offset. An offset is assigned to each element in an array and is used to reference to it.
As in C or Java, array indexing begins at zero. A negative index is assumed relative to the array's end —- that is, an index of -1 denotes the array's last element, -2 indicates the array's next to last element, and so on. Ruby arrays can hold objects such as String, Integer, Fix num, Hash, Symbol, even other Array objects. Ruby arrays are not as rigid as arrays in other languages. Ruby arrays grow automatically while adding elements to them.

Ex:

```
names = Array.new(20)
puts names.size # This returns 20
puts names.length # This also returns 20
```

FEATURES

2. METHODS:

Methods in Ruby are a lot like functions in other programming languages. Ruby methods are used to group together one or more recurring statements.
The name of the method should start with a lowercase letter. If you start a method name with an uppercase letter, Ruby may mistake it for a constant and wrongly process the call.
Methods should be declared before they are called; otherwise, Ruby will throw an exception for executing an undefined method.
Ex:

```
def test(a1 = "Ruby", a2 = "Perl")
puts "The programming language is #{a1}"
puts "The programming language is #{a2}"
end
test "C", "C++"
test
```

This will produce the following result –

```
The programming language is C
The programming language is C++
The programming language is Ruby
The programming language is Perl
```

The initialize method is a common Ruby class method that functions in a similar fashion to other object-oriented programming languages' constructors. When you wish to initialize some class variables at the moment of object creation, the initialize function comes in handy. This method may take a list of parameters, and it would be prefixed by the def keyword, just like any other ruby method, as seen below.

```
class Box
 def initialize(w,h)
@width, @height = w, h
end
end
```

FEATURES

3. OBJECT-ORIENTED:

Ruby is an object-oriented programming language, which means that everything appears to Ruby as an object. Even the most basic values in Ruby, such as characters, integers, and even true and false, are objects. An object that is an instance of the Class is even a class. This chapter will walk you through all of the key features of Object-Oriented Ruby.

A class specifies an object's form by combining data representation and methods for modifying that data into a single package. Members of a class are the data and methods that make up the class.

When you create a class, you're essentially creating a blueprint for a data type. This doesn't describe any data, but it does define what the class name signifies, that is, what a class object will be made up of and what actions can be done on it. A class definition begins with the keyword class, then the class name, and ends with a period. For example, using the keyword class, we defined the Box class as follows:

```
class Box
    Code
end
```

The name must begin with a capital letter, and names that comprise more than one word are run together with each word capitalised and no separating letters, according to tradition (CamelCase).

A class serves as the blueprint for things; therefore, an object is essentially formed from one. The new keyword is used to declare class objects. The following statements declare two Box objects.

```
box1 = Box.new
box2 = Box.new
```

FEATURES

4. EXCEPTION:

The execution and the exception are inextricably linked. If you open a file that does not exist and do not handle the issue correctly, your application is deemed to be of poor quality.

If an exception occurs, the program will terminate. As a result, exceptions are used to manage various types of problems that may arise during the execution of a program and take necessary action rather than stopping the operation entirely.

Ruby has a handy framework for dealing with exceptions. We use rescue clauses to inform Ruby what sorts of exceptions we want to manage, and we wrap the code that may cause an exception in a begin/end block.

Syntax:

```
begin
# -
 rescue OneTypeOfException
 # -
rescue AnotherTypeOfException
# -
else
#
Other exceptions
ensure
# Always will be executed
end
```

Everything is safe, from the beginning to the end. Control is given to the block between rescue and end if an exception occurs during the execution of this block of code.

Ruby compares the raised Exception to each of the arguments in turn for each rescue clause in the begin block. If the exception listed in the rescue clause is the same type as the presently thrown exception, or is a superclass of that exception, the match will succeed.

We are permitted to apply an else clause after all the rescue clauses if an exception does not match any of the error types supplied.

```
begin
file = open("/unexistant_file")
if file
puts "File opened successfully"
End
 rescue
file = STDIN
end
print file, "==", STDIN, "\n"
```

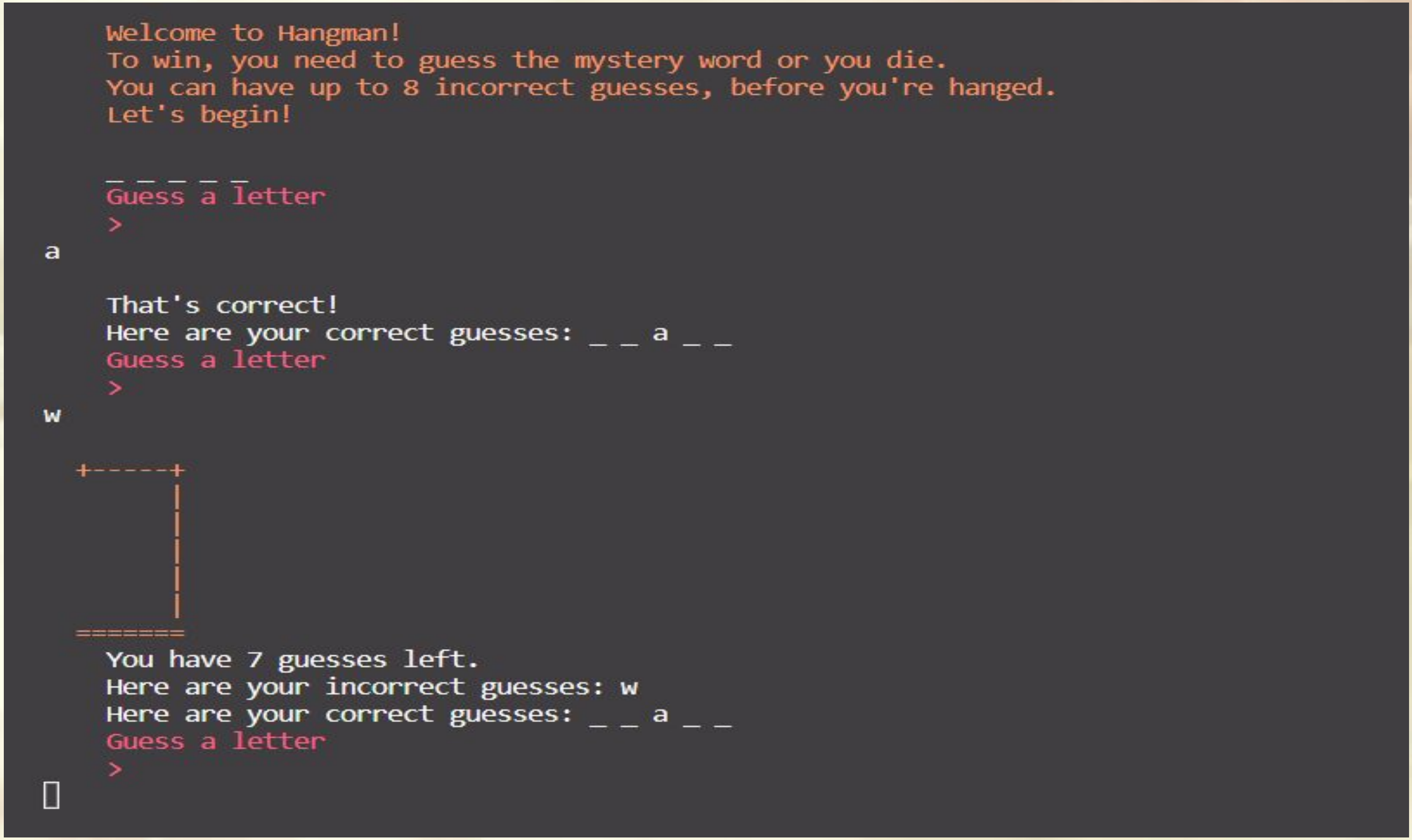
This will produce the following result. You can see that STDIN is substituted to file because open failed.

Introduction

User would get 8 chance to guess the word , every wrong letter guess , will lose one chance and every right letter guess , will get the message that would guess the right word.

Every time user will see the Hangman movement according to words guess.

When user guess correct word

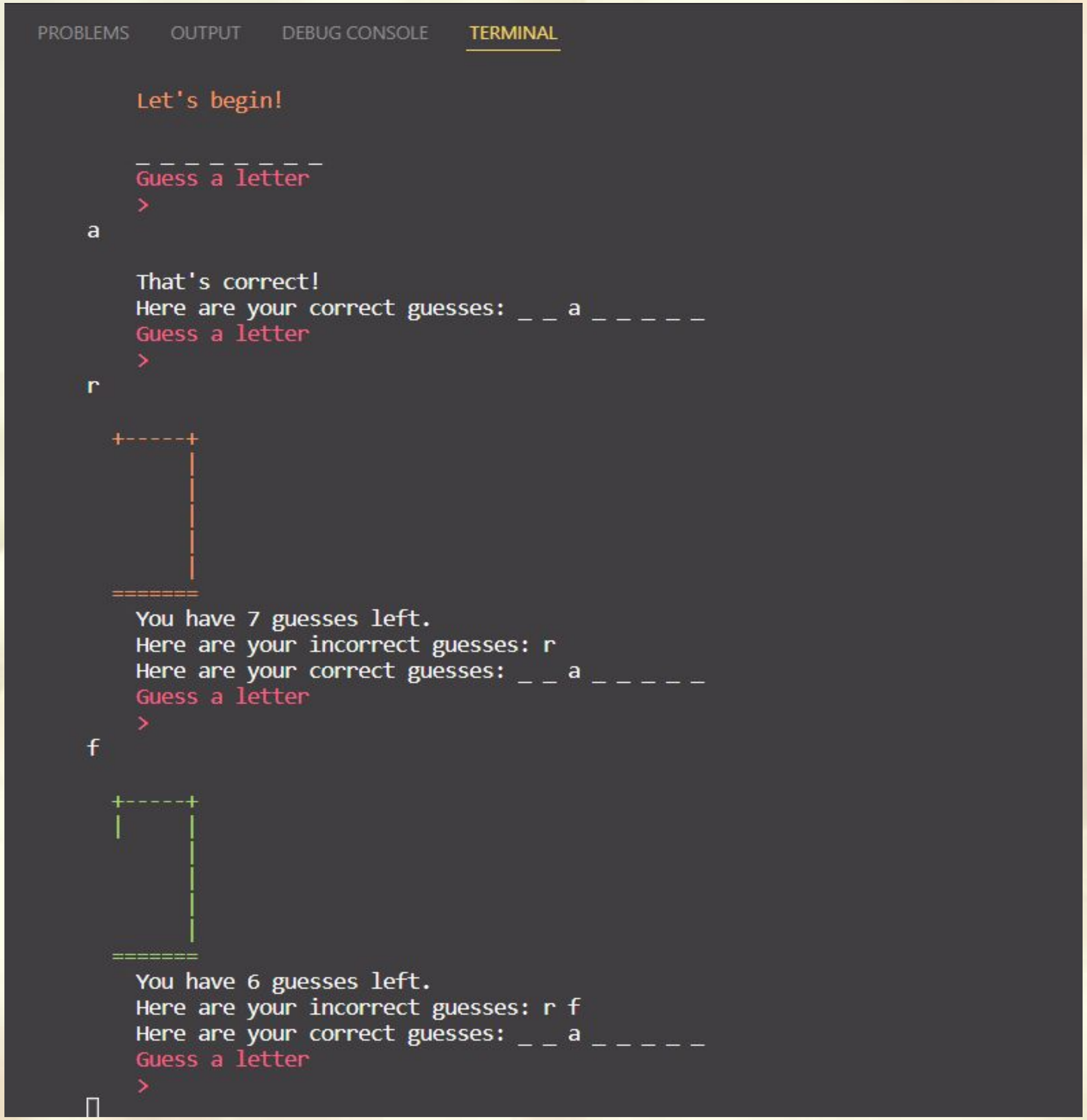


```
    Welcome to Hangman!
    To win, you need to guess the mystery word or you die.
    You can have up to 8 incorrect guesses, before you're hanged.
    Let's begin!


    _ _ _ _ _
    Guess a letter
    >
a

    That's correct!
    Here are your correct guesses: _ _ a _ _
    Guess a letter
    >
w

  +------+
         |
         |
         |
         |
         |
         |
  =======
    You have 7 guesses left.
    Here are your incorrect guesses: w
    Here are your correct guesses: _ _ a _ _
    Guess a letter
    >
▯
```



```
    Let's begin!

    _ _ _ _ _ _ _ _
    Guess a letter
    >
a

    That's correct!
    Here are your correct guesses: _ _ a _ _ _ _ _
    Guess a letter
    >
r

  +-----+
        |
        |
        |
        |
        |
  =======
    You have 7 guesses left.
    Here are your incorrect guesses: r
    Here are your correct guesses: _ _ a _ _ _ _ _
    Guess a letter
    >
f

  +-----+
  |     |
        |
        |
        |
        |
  =======
    You have 6 guesses left.
    Here are your incorrect guesses: r f
    Here are your correct guesses: _ _ a _ _ _ _ _
    Guess a letter
    >
```

When user guess wrong word

Motivation

- People can learn new words through playing.
- Also helpful for remembering the spelling of new words.

Things I have learned

- Ruby language , logic , syntax .
- Using branches in Git
- Methods call and compile the files.

Things I have struggled with

- Testing output with RSpec.
- Scope of instance variables.
- Knowing when to switch off and take breaks.

THANK YOU