Burger Game
Ruby Terminal App

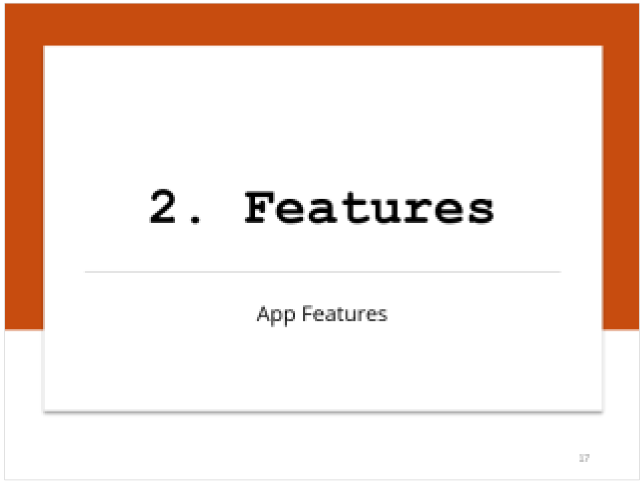
Walk-Through & Review of Terminal App

Table of Contents

1. App Overview

Flowchart, Logic Flow and Structure

3


2. Features

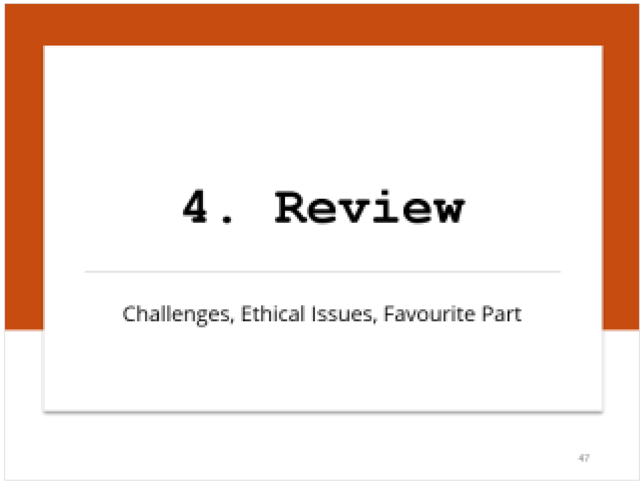
App Features

17

3. Code Overview

App Code Overview

28


4. Review

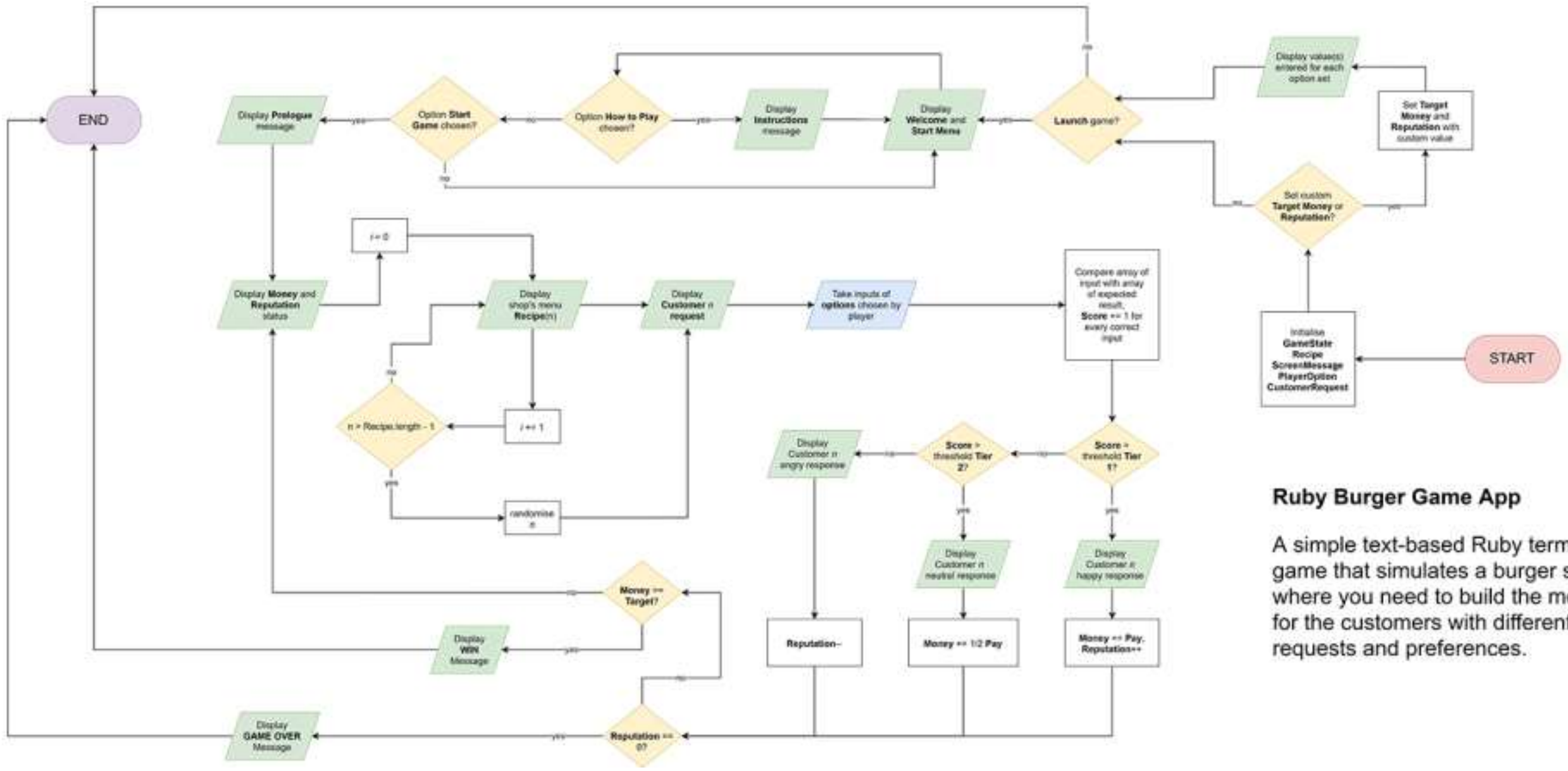
Challenges, Ethical Issues, Favourite Part

47

1. App Overview

Flowchart, Logic Flow and Structure

Flowchart

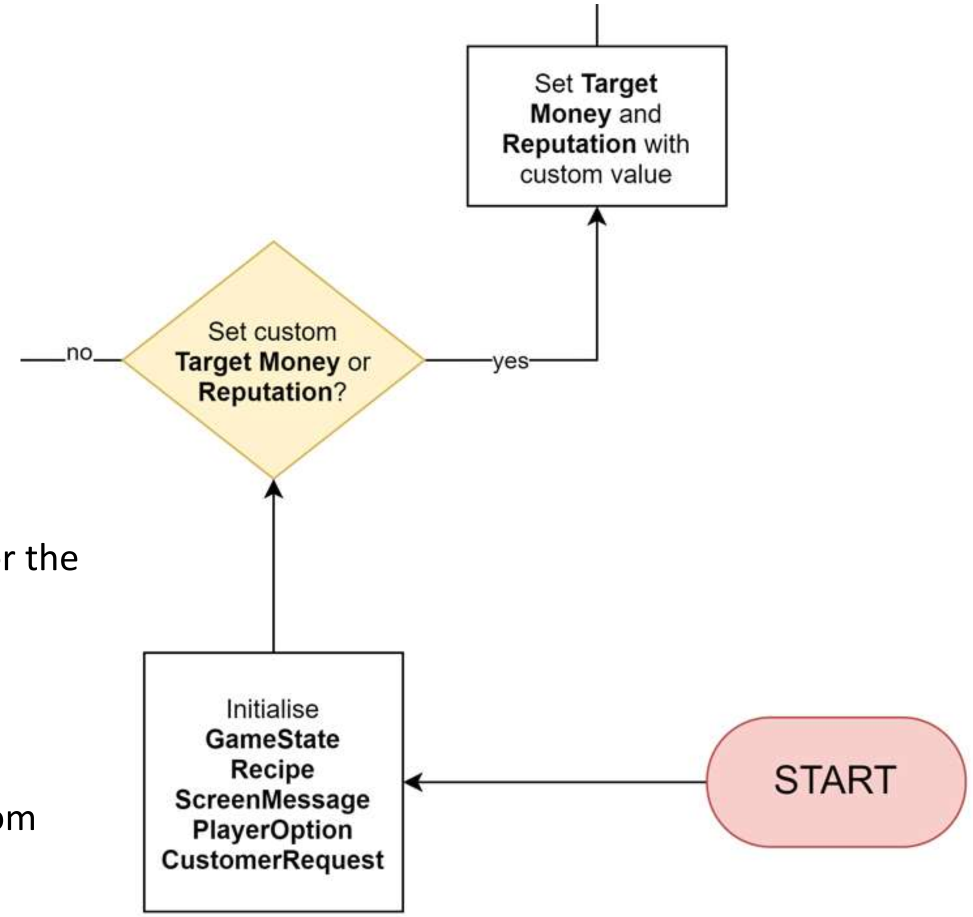

START
Initialise GameState Recipe ScreenMessage PlayerOption CustomerRequest
Set custom Target Money or Reputation?
yes
no
Set Target Money and Reputation with custom value
Display value(s) entered for each option set
Launch game?
yes
no
Display Welcome and Start Menu
Option How to Play chosen?
yes
no
Display Instructions message
Option Start Game chosen?
yes
no
Display Prologue message
END
Display Money and Reputation status
i = 0
Display shop's menu Recipe(n)
i += 1
n > Recipe.length - 1
yes
no
randomise n
Display Customer n request
Take inputs of options chosen by player
Compare array of input with array of expected result, Score += 1 for every correct input
Score > threshold Tier 1?
yes
no
Display Customer n happy response
Money += Pay, Reputation++
Score > threshold Tier 2?
yes
no
Display Customer n neutral response
Money += 1/2 Pay
Display Customer n angry response
Reputation--
Reputation == 0?
yes
no
Display GAME OVER Message
Money == Target?
yes
no
Display WIN Message

Ruby Burger Game App

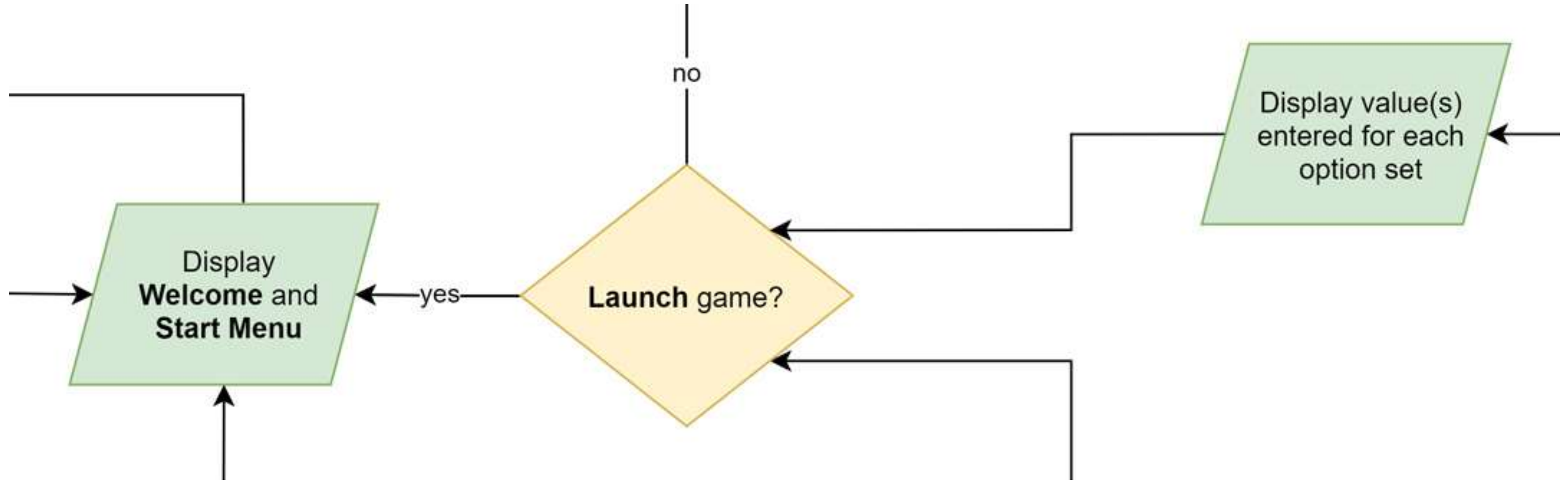
A simple text-based Ruby terminal game that simulates a burger shop, where you need to build the meal for the customers with different requests and preferences.

Logic Flow

- **Initialisation**:
  - Create instances, initialise variables with starting values within the app.
- **Command line** options:
  - ‘h’ or ‘--h’: Display Help menu for the app.
  - ‘-m’ or ‘--money’: Set custom target money goal.
  - ‘-r’ or ‘--reputation’: Set custom maximum reputation.


START
Initialise GameState Recipe ScreenMessage PlayerOption CustomerRequest
Set custom Target Money or Reputation?
no
yes
Set Target Money and Reputation with custom value

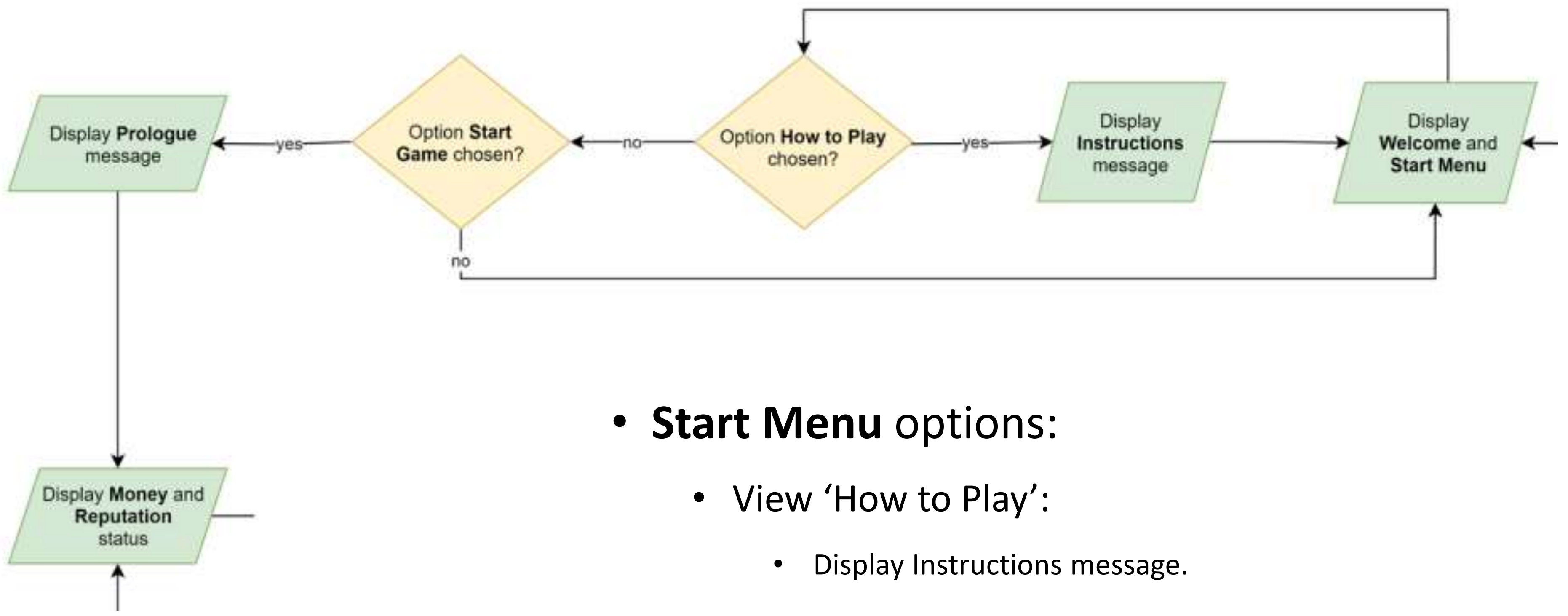
Logic Flow


no
Display value(s) entered for each option set
Display Welcome and Start Menu
yes
Launch game?

- **Display** input value(s) if setting custom money/reputation.

- Options to **launch** the game (or **exit** the app):
  - Display Welcome message.
  - Display Start Menu.

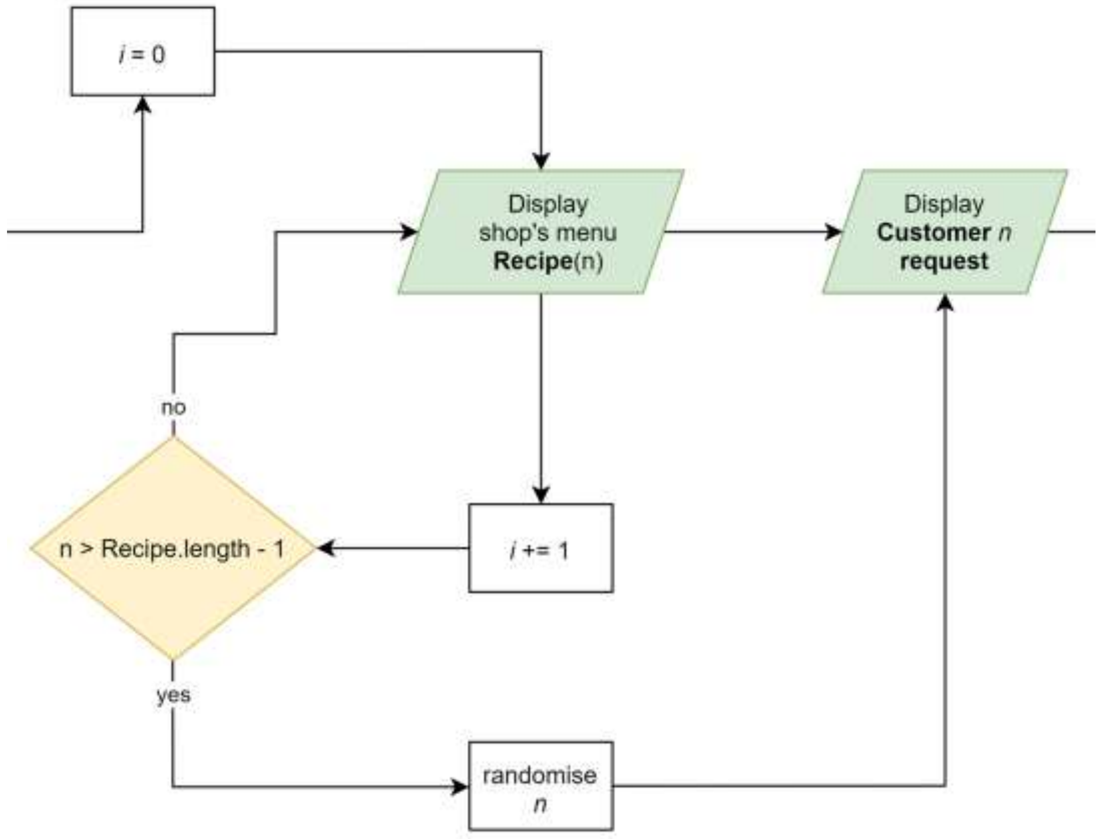
Logic Flow



Display Prologue message
yes
Option Start Game chosen?
no
Option How to Play chosen?
yes
Display Instructions message
Display Welcome and Start Menu
no
Display Money and Reputation status

- **Start Menu** options:
  - View ‘How to Play’:
    - Display Instructions message.
  - Start Game:
    - Display Prologue message
    - Display of starting money and reputation status.

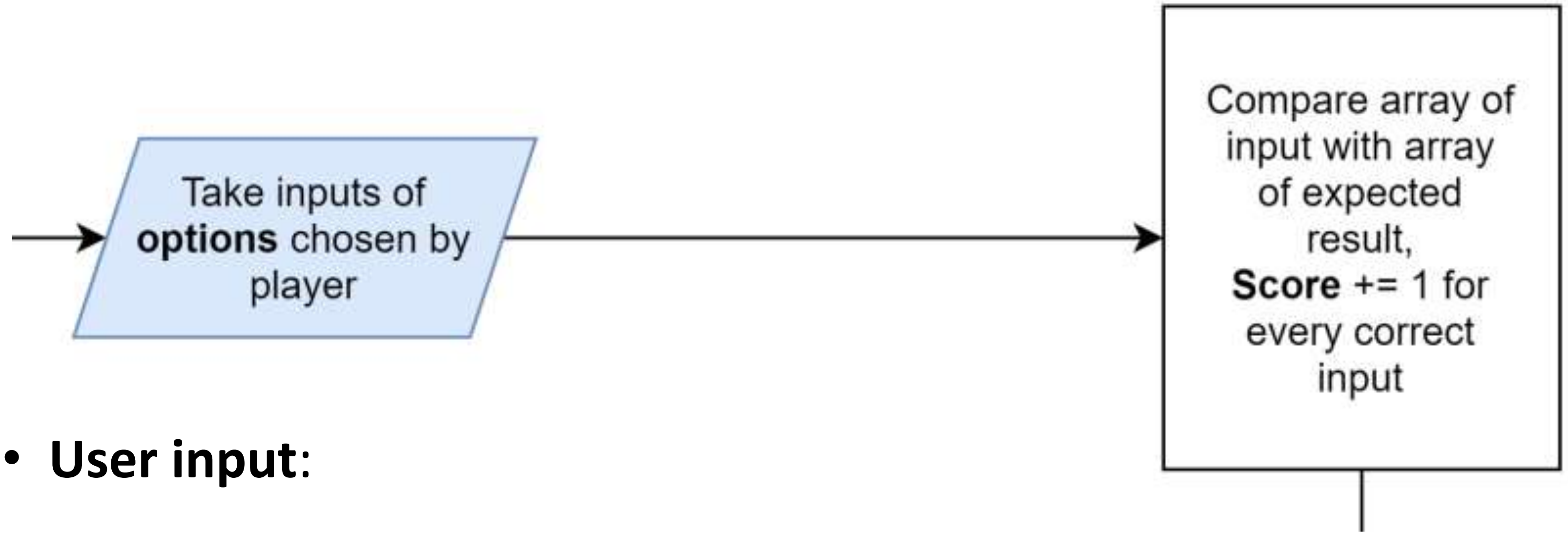
Logic Flow



i = 0
Display shop's menu **Recipe**(n)
Display **Customer** *n* **request**
no
n > Recipe.length - 1
i += 1
yes
randomise *n*

- Display shop's menu **Recipes**:
  - Loop to display all recipes
- Display customer request **randomly**:
  - Take random customer *n* response to display

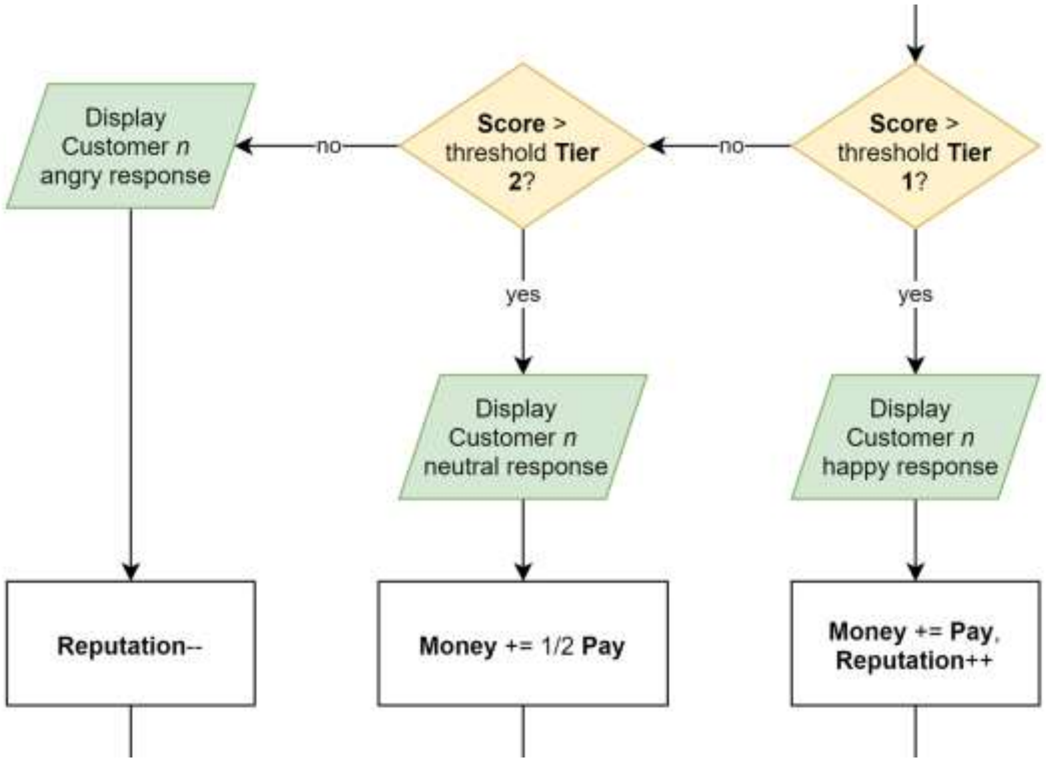
Logic Flow


Take inputs of **options** chosen by player
Compare array of input with array of expected result, **Score** += 1 for every correct input

- **User input**:
  - Selectable options: List of ingredients.
  - Text input: Prompt to enter number 0 – 5.

- **Compare** user input with expected result:
  - Calculate score: Add +1 to score for every correct input

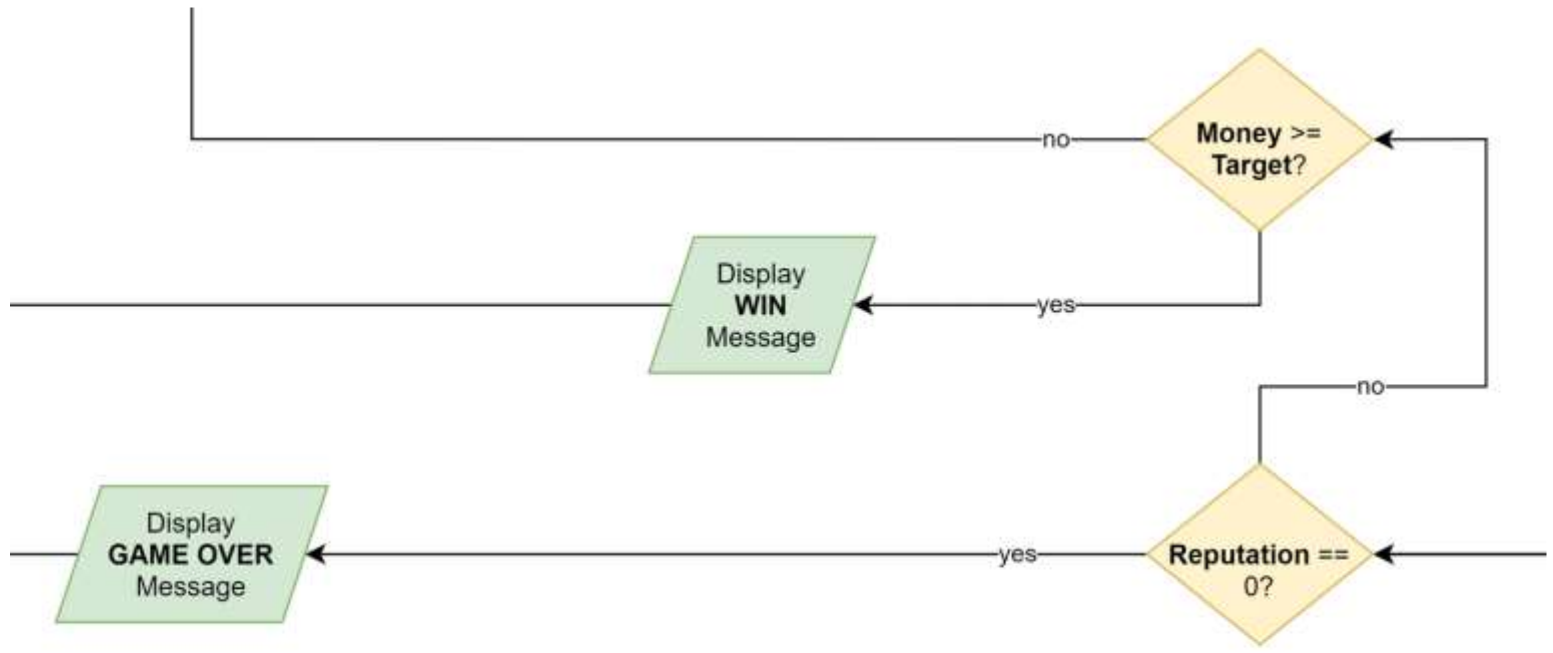
Logic Flow



Score > threshold Tier 1?
no
Score > threshold Tier 2?
no
Display Customer *n* angry response
yes
Display Customer *n* neutral response
yes
Display Customer *n* happy response
Reputation--
Money += 1/2 Pay
Money += Pay, Reputation++

- Display **customer response**:
  - Threshold is pre-set in the app
  - Show appropriate response based on score
  - Add/subtract points

Logic Flow



- Win/Game Over **condition**:
  - Win: current money >= target money
    - Display Win message (ASCII art)
  - Game Over: current reputation == 0
    - Display Game Over message

Not met?

Back to display shop's menu and continue to random customer's request, prompt input, calculate, display response, then Win/Game Over condition again.

Structure

The app has several classes, with methods exclusively available within them.

a) **GameState**

+ Attributes:

- Target money (float)
- Max reputation (int)
- Payment (float)
- Current money (float)
- Current reputation (int)

+ Actions:

- Set target money
- Set max reputation
- Update money
- Update reputation
- Display game state

Structure (cont.)

b) **ScreenMessage**

+ Attributes:

- Title (string)
- Message (string)

+ Actions:

- Display welcome
- Display instructions
- Display prologue
- Display Win message
- Display Game Over message

c) **Recipe**

+ Attributes:

- Recipe name (string)
- Ingredient-quantity list (hash)

+ Actions:

- Display recipe

Structure (cont.)

d) **CustomerRequest**

+ Attributes:

- Customer name (string)
- Request text (string)
- Recipe name (string)
- Ingredient change (hash)
- Request text (string)
- Response (string)

+ Actions:

- Get request
- Display request
- Display response

e) **PlayerOption**

+ Attributes:

- User input (string)

+ Actions:

- Launch game
- Start game
- Get selection

Structure (cont.)

f) **ScoreComparison**

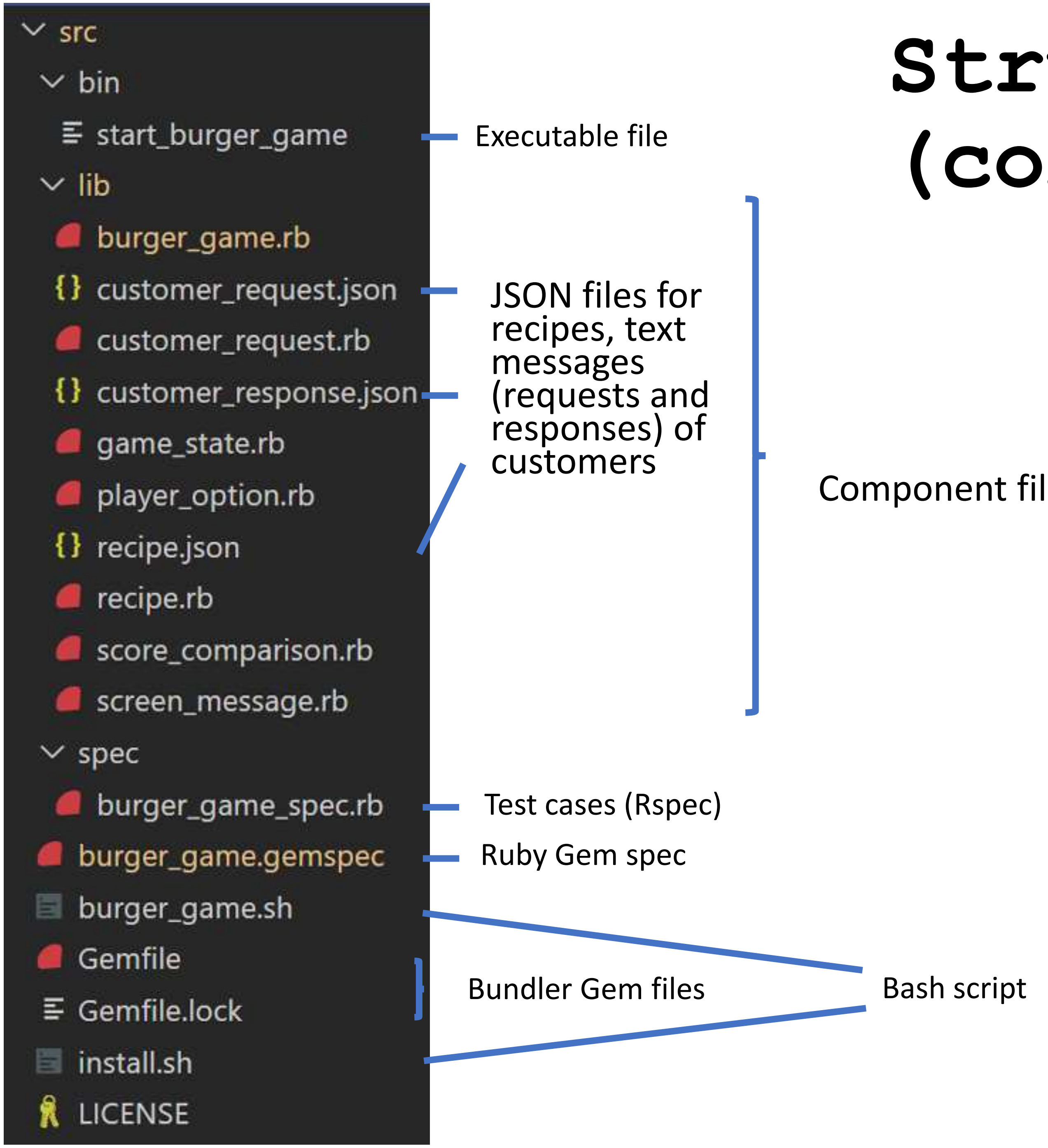
+ Attributes:

- Max score (int)
- Threshold (int)
- Player recipe (hash)
- Customer recipe (hash)

+ Actions:

- Get score
- Get mood (customer)
- Calculate state (update)

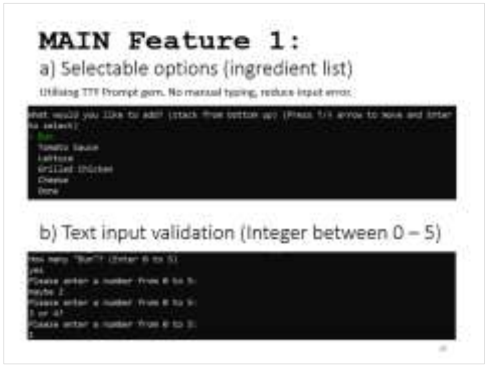
Structure (cont.)



```
src
  bin
    start_burger_game
  lib
    burger_game.rb
    customer_request.json
    customer_request.rb
    customer_response.json
    game_state.rb
    player_option.rb
    recipe.json
    recipe.rb
    score_comparison.rb
    screen_message.rb
  spec
    burger_game_spec.rb
  burger_game.gemspec
  burger_game.sh
  Gemfile
  Gemfile.lock
  install.sh
  LICENSE
```

2. Features

App Features



MAIN Feature 1:

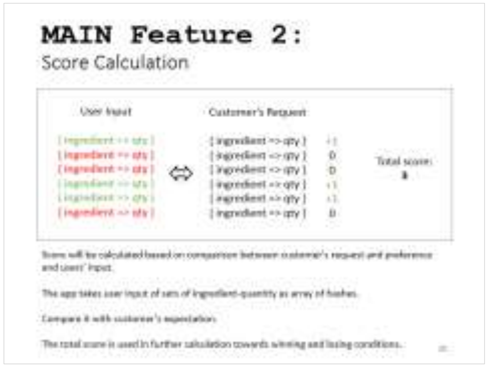
a) Selectable options (ingredient list)

Utilising TTY Prompt gem. No manual typing, reduce input error.

```
What would you like to add? (stack from bottom up) (Press ↑/↓ arrow to move and Enter to select)
‣ Bun
  Tomato Sauce
  Lettuce
  Grilled Chicken
  Cheese
  Done
```

b) Text input validation (Integer between 0 – 5)

```
How many "Bun"? (Enter 0 to 5)
yes
Please enter a number from 0 to 5:
maybe 2
Please enter a number from 0 to 5:
3 or 4?
Please enter a number from 0 to 5:
I
```



MAIN Feature 2:

Score Calculation

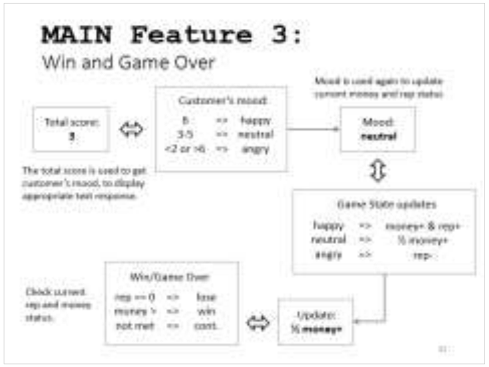
User Input		Customer's Request		
[ingredient => qty]	⇔	[ingredient => qty]	+1	Total score: 3
[ingredient => qty]		[ingredient => qty]	0	
[ingredient => qty]		[ingredient => qty]	0	
[ingredient => qty]		[ingredient => qty]	+1	
[ingredient => qty]		[ingredient => qty]	+1	
[ingredient => qty]		[ingredient => qty]	0	

Score will be calculated based on comparison between customer's request and preference and users' input.

The app takes user input of sets of ingredient-quantity as array of hashes.

Compare it with customer's expectation.

The total score is used in further calculation towards winning and losing conditions.



MAIN Feature 3:

Win and Game Over

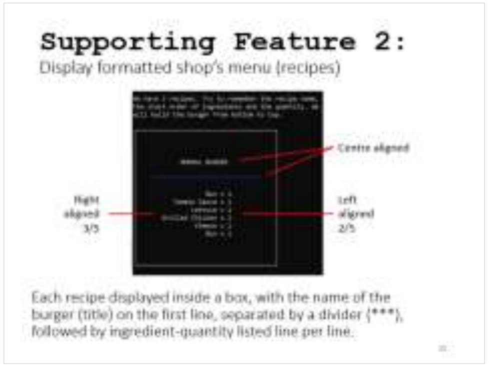
The total score is used to get customer's mood, to display appropriate text response.

Supporting Feature 1:

Options to view instructions or to start the game

How to Play
Start Game

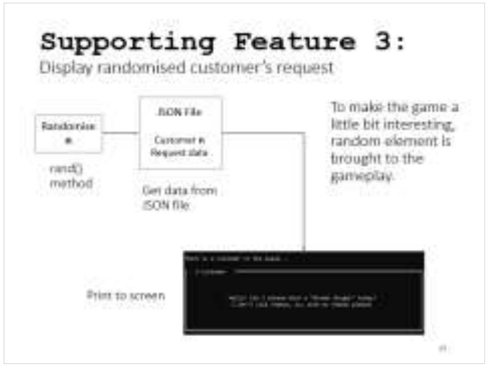
After Welcome message, menu:
- *How to Play*: view instructions, back to start menu
- *Start Game*: start game and continue to the next scene



Supporting Feature 2:

Display formatted shop's menu (recipes)

Each recipe displayed inside a box, with the name of the burger (title) on the first line, separated by a divider (***), followed by ingredient-quantity listed line per line.



Supporting Feature 3:

Display randomised customer's request

To make the game a little bit interesting, random element is brought to the gameplay.

Supporting Feature 4:

Display various customers' responses

Happy
Neutral
Angry

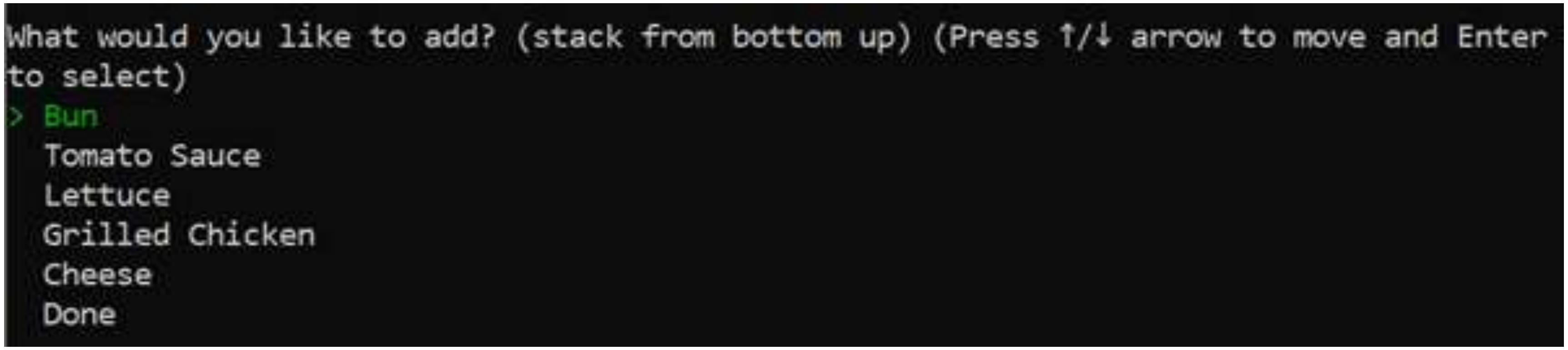
The app will show the corresponding customer's response based on mood calculation.

Features

MAIN Feature 1:

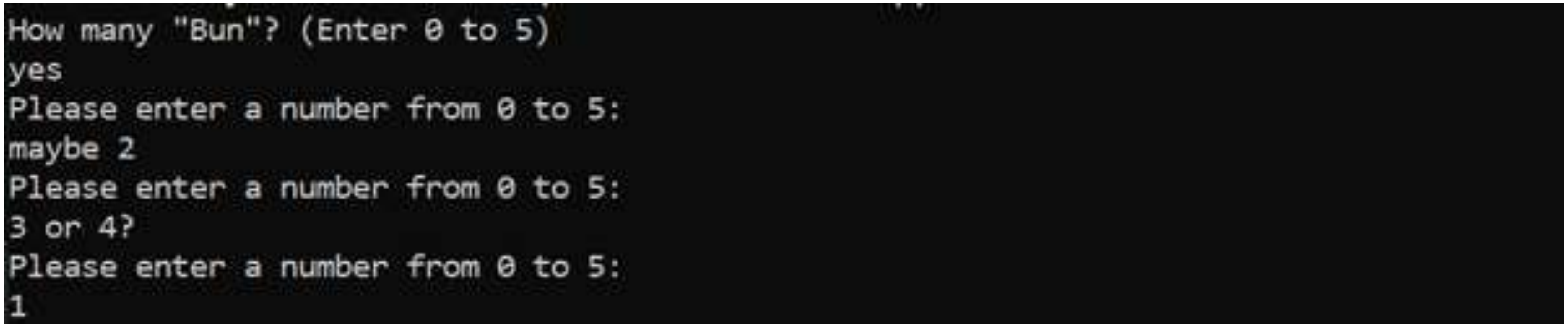
a) Selectable options (ingredient list)

Utilising TTY Prompt gem. No manual typing, reduce input error.



```
What would you like to add? (stack from bottom up) (Press ↑/↓ arrow to move and Enter
to select)
> Bun
  Tomato Sauce
  Lettuce
  Grilled Chicken
  Cheese
  Done
```

b) Text input validation (Integer between 0 – 5)



```
How many "Bun"? (Enter 0 to 5)
yes
Please enter a number from 0 to 5:
maybe 2
Please enter a number from 0 to 5:
3 or 4?
Please enter a number from 0 to 5:
1
```

MAIN Feature 2:

Score Calculation

User Input		Customer's Request		
{ ingredient => qty }		{ ingredient => qty }	+1	
{ ingredient => qty }		{ ingredient => qty }	0	Total score:
{ ingredient => qty }	⇔	{ ingredient => qty }	0	**3**
{ ingredient => qty }		{ ingredient => qty }	+1	
{ ingredient => qty }		{ ingredient => qty }	+1	
{ ingredient => qty }		{ ingredient => qty }	0	

Score will be calculated based on comparison between customer's request and preference and users' input.

The app takes user input of sets of ingredient-quantity as array of hashes.

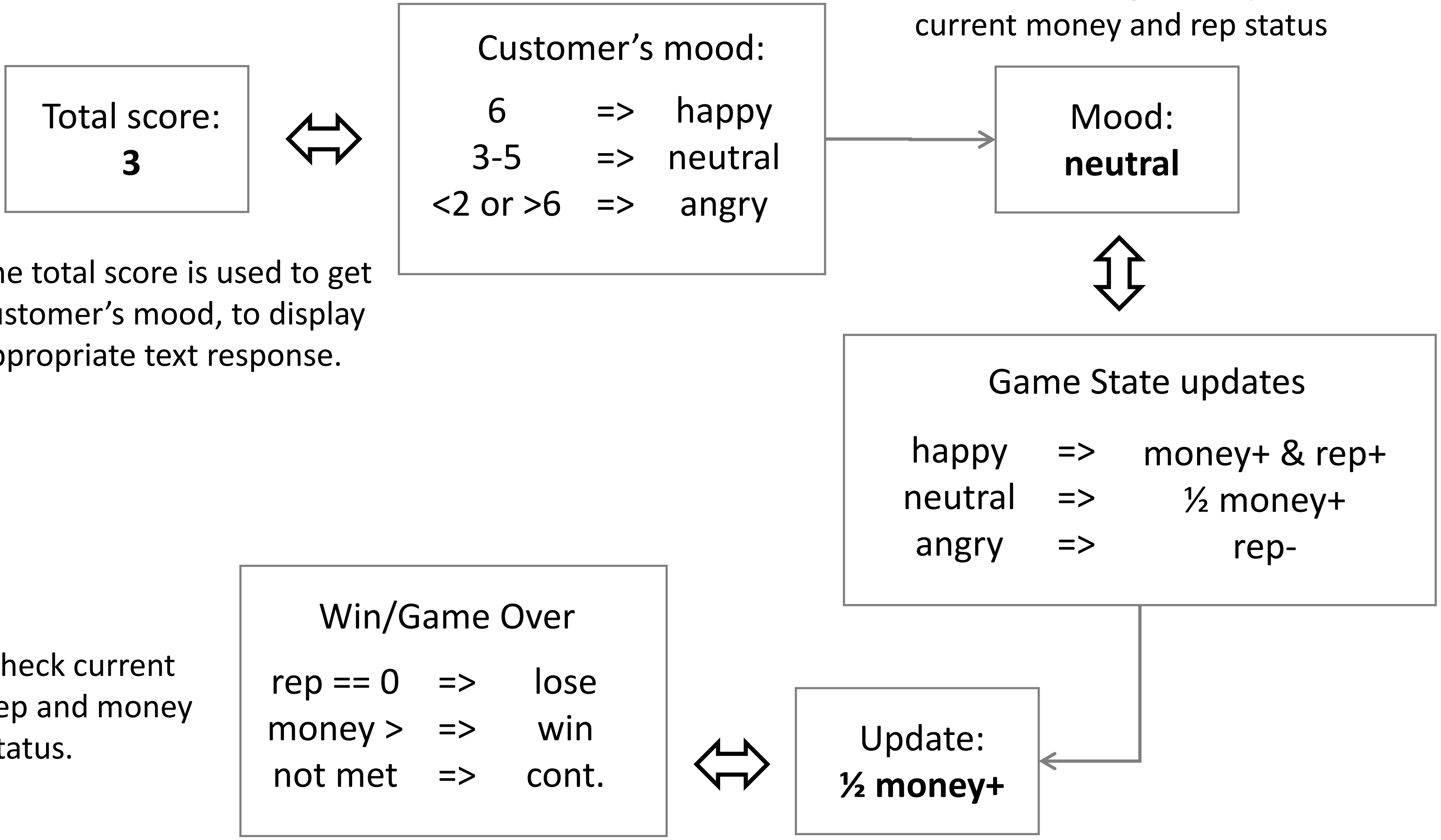
Compare it with customer's expectation.

The total score is used in further calculation towards winning and losing conditions.

MAIN Feature 3:

Win and Game Over

Mood is used again to update current money and rep status



Customer's mood:

6	=>	happy
3-5	=>	neutral
<2 or >6	=>	angry

The total score is used to get customer's mood, to display appropriate text response.

Game State updates

happy	=>	money+ & rep+
neutral	=>	½ money+
angry	=>	rep-

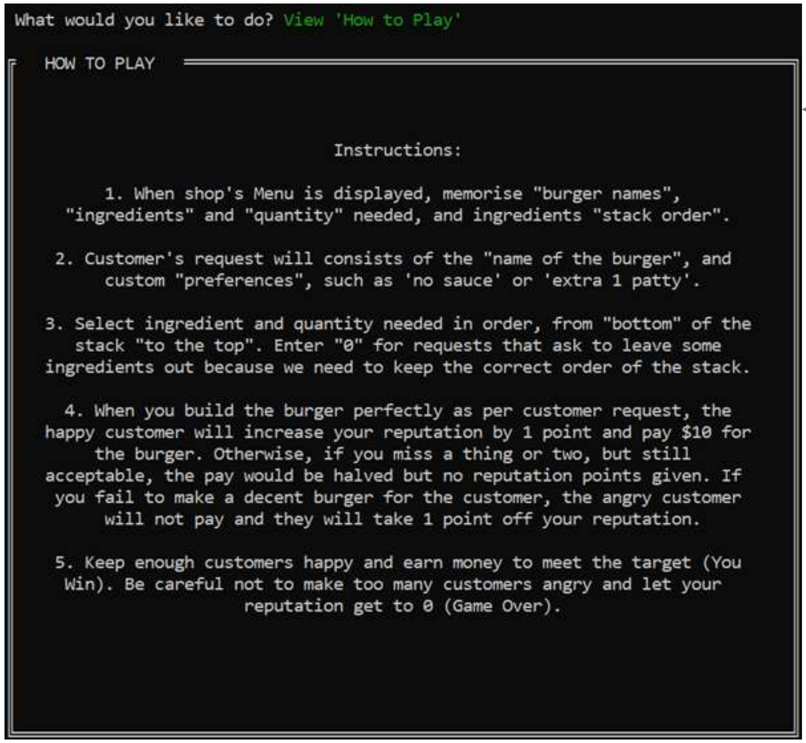
Check current rep and money status.

Win/Game Over

rep == 0	=>	lose
money >	=>	win
not met	=>	cont.

Supporting Feature 1:

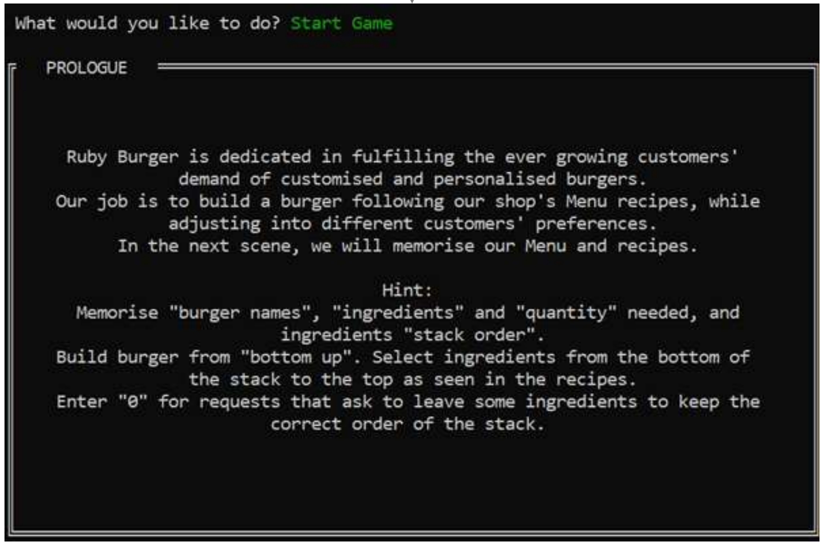
Options to view instructions or to start the game



```
What would you like to do? View 'How to Play'

  HOW TO PLAY

                              Instructions:

          1. When shop's Menu is displayed, memorise "burger names",
       "ingredients" and "quantity" needed, and ingredients "stack order".

        2. Customer's request will consists of the "name of the burger", and
           custom "preferences", such as 'no sauce' or 'extra 1 patty'.

       3. Select ingredient and quantity needed in order, from "bottom" of the
         stack "to the top". Enter "0" for requests that ask to leave some
       ingredients out because we need to keep the correct order of the stack.

         4. When you build the burger perfectly as per customer request, the
       happy customer will increase your reputation by 1 point and pay $10 for
          the burger. Otherwise, if you miss a thing or two, but still
       acceptable, the pay would be halved but no reputation points given. If
        you fail to make a decent burger for the customer, the angry customer
           will not pay and they will take 1 point off your reputation.

        5. Keep enough customers happy and earn money to meet the target (You
         Win). Be careful not to make too many customers angry and let your
                      reputation get to 0 (Game Over).
```

How to Play

Start Game



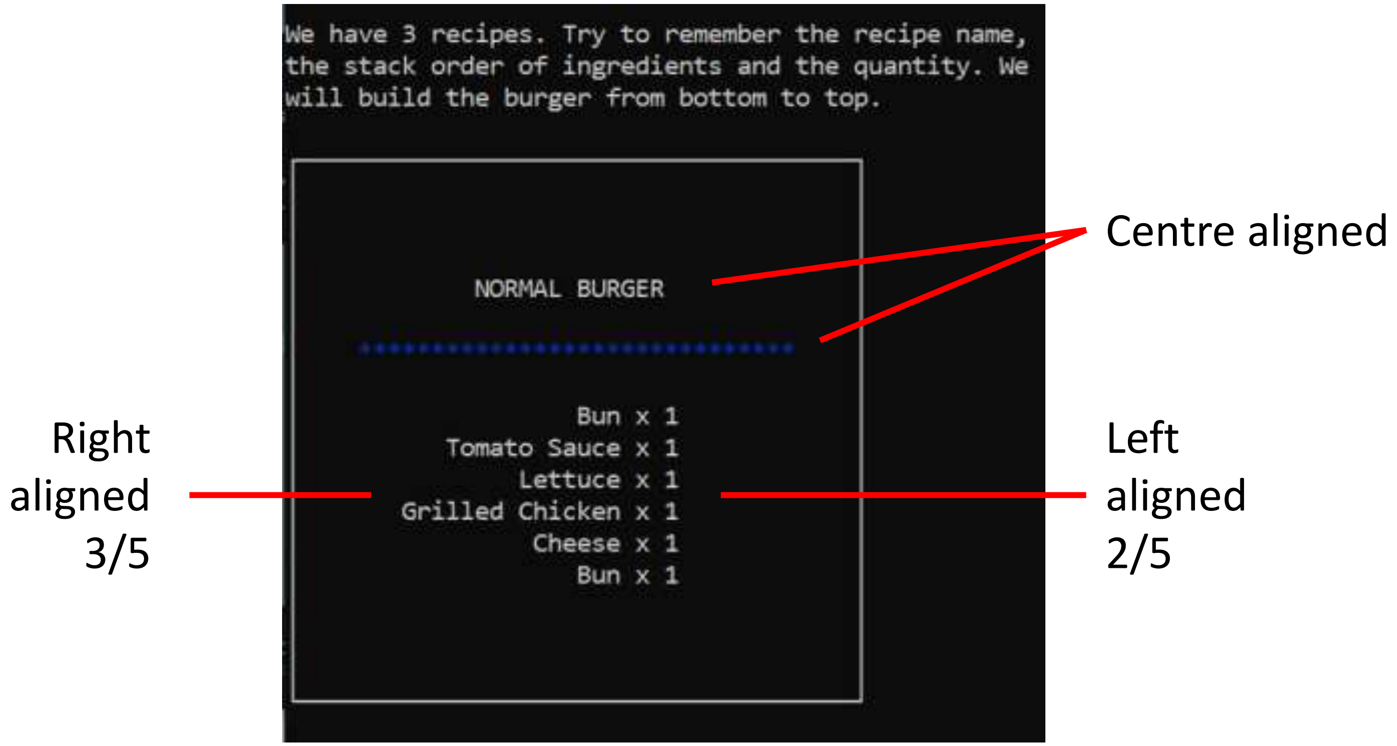
```
What would you like to do? Start Game

  PROLOGUE

        Ruby Burger is dedicated in fulfilling the ever growing customers'
                  demand of customised and personalised burgers.
       Our job is to build a burger following our shop's Menu recipes, while
                  adjusting into different customers' preferences.
            In the next scene, we will memorise our Menu and recipes.

                                    Hint:
         Memorise "burger names", "ingredients" and "quantity" needed, and
                         ingredients "stack order".
       Build burger from "bottom up". Select ingredients from the bottom of
                    the stack to the top as seen in the recipes.
       Enter "0" for requests that ask to leave some ingredients to keep the
                          correct order of the stack.
```

After Welcome message, menu:

- *How to Play*: view instructions, back to start menu
- *Start Game*: start game and continue to the next scene

Supporting Feature 2:

Display formatted shop's menu (recipes)

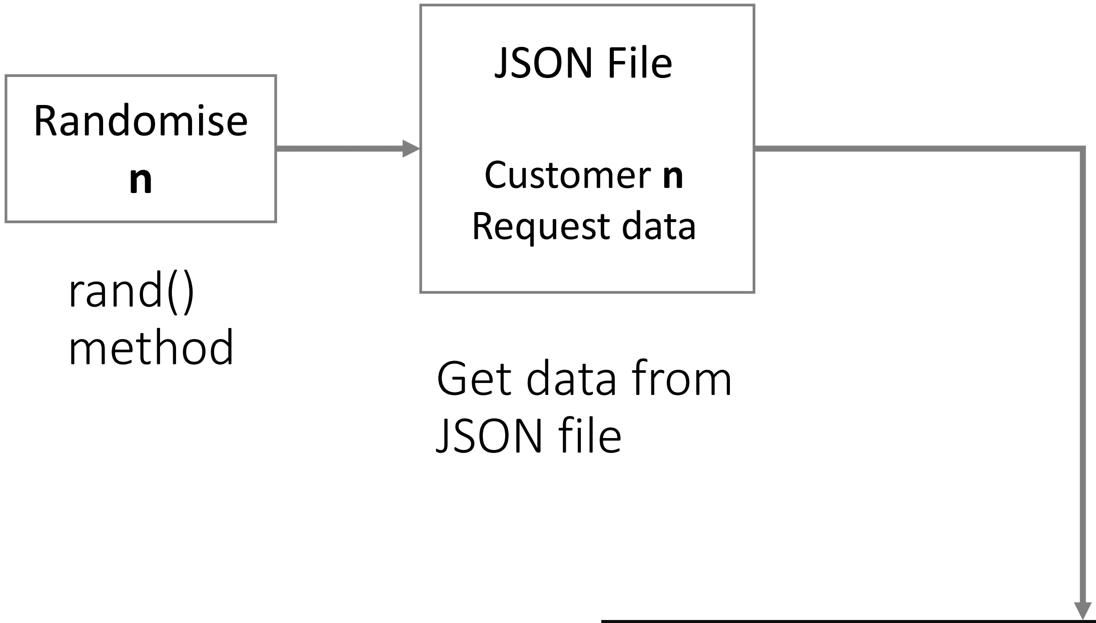


```
We have 3 recipes. Try to remember the recipe name,
the stack order of ingredients and the quantity. We
will build the burger from bottom to top.

 ______________________________________
|                                      |
|                                      |
|            NORMAL BURGER             |
|                                      |
|    ******************************    |
|                                      |
|                Bun x 1               |
|       Tomato Sauce x 1               |
|            Lettuce x 1               |
|    Grilled Chicken x 1               |
|             Cheese x 1               |
|                Bun x 1               |
|                                      |
|______________________________________|
```

Each recipe displayed inside a box, with the name of the burger (title) on the first line, separated by a divider (***), followed by ingredient-quantity listed line per line.

Supporting Feature 3:

Display randomised customer’s request

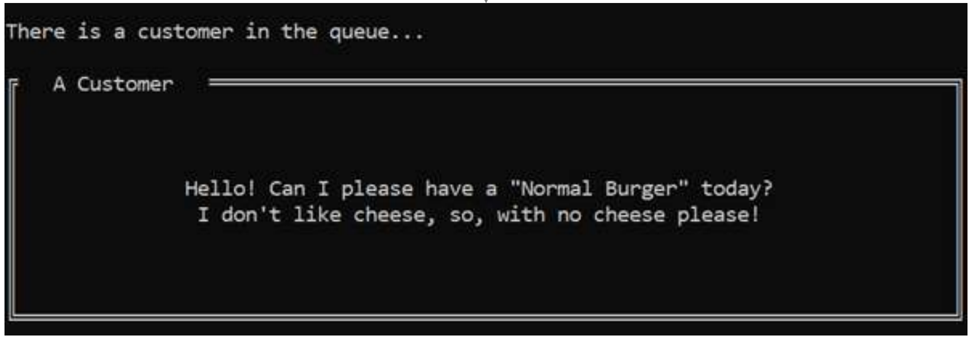


rand() method

Get data from JSON file

To make the game a little bit interesting, random element is brought to the gameplay.

Print to screen



```
There is a customer in the queue...

╔ A Customer ═══════════════════════════════════════════════╗


        Hello! Can I please have a "Normal Burger" today?
          I don't like cheese, so, with no cheese please!

╚═══════════════════════════════════════════════════════════╝
```

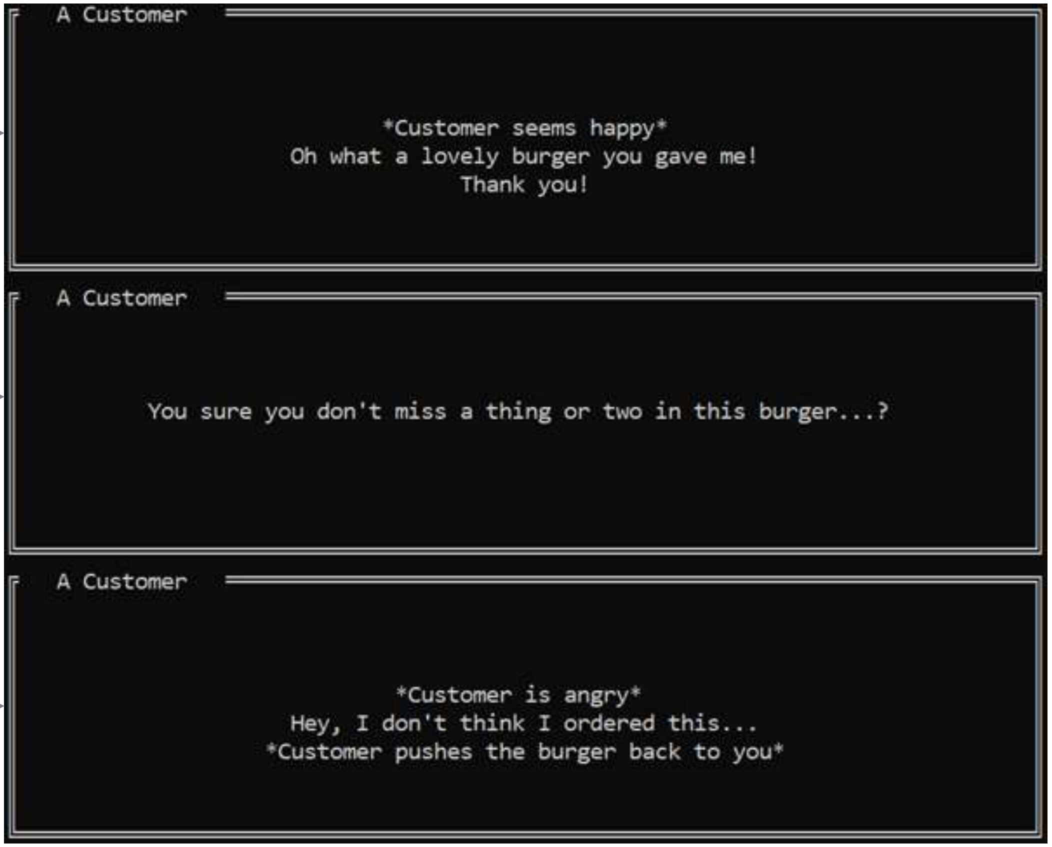
Supporting Feature 4:

Display various customers' responses

Happy

Neutral

Angry



```
A Customer

*Customer seems happy*
Oh what a lovely burger you gave me!
Thank you!
```

```
A Customer

You sure you don't miss a thing or two in this burger...?
```

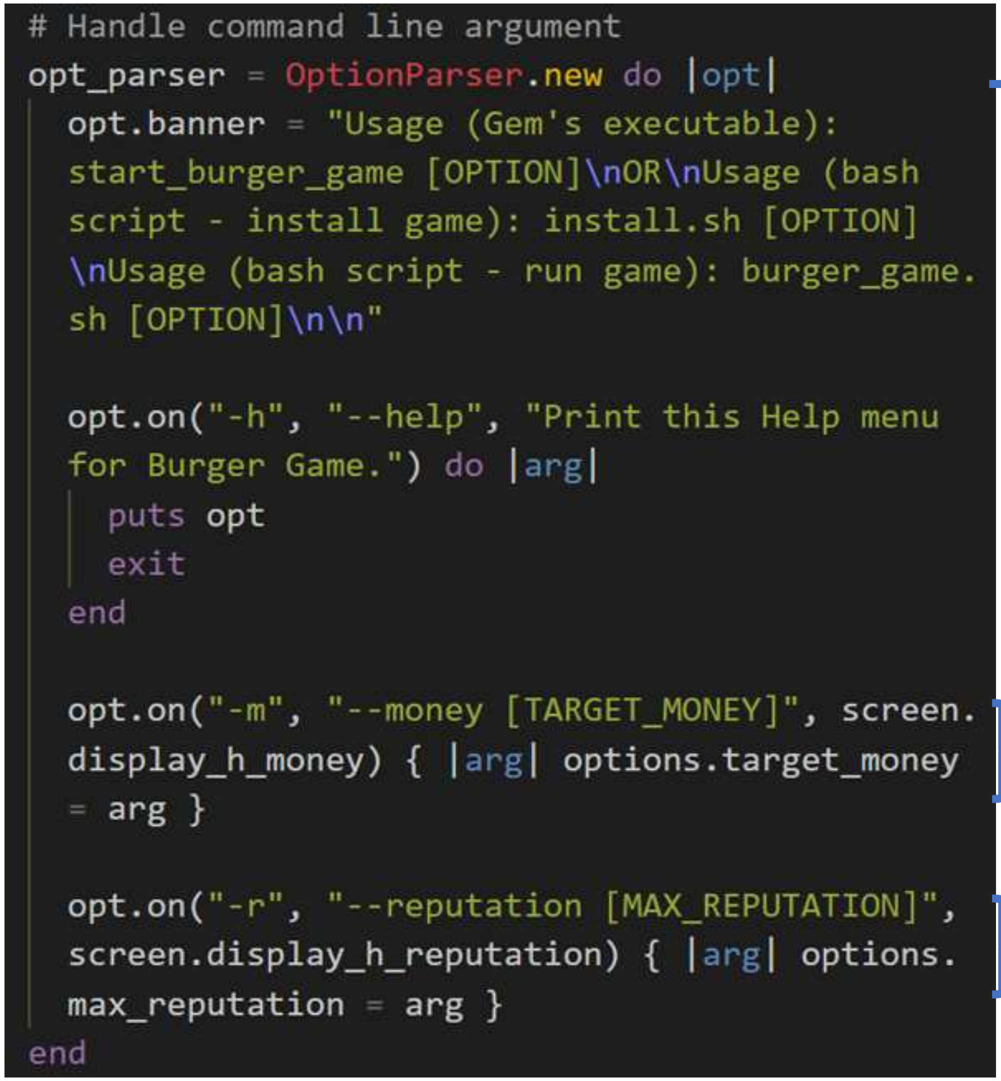
```
A Customer

*Customer is angry*
Hey, I don't think I ordered this...
*Customer pushes the burger back to you*
```

The app will show the corresponding customer's response based on mood calculation.

3. Code Overview

App Code Overview

Command Line Argument(s)



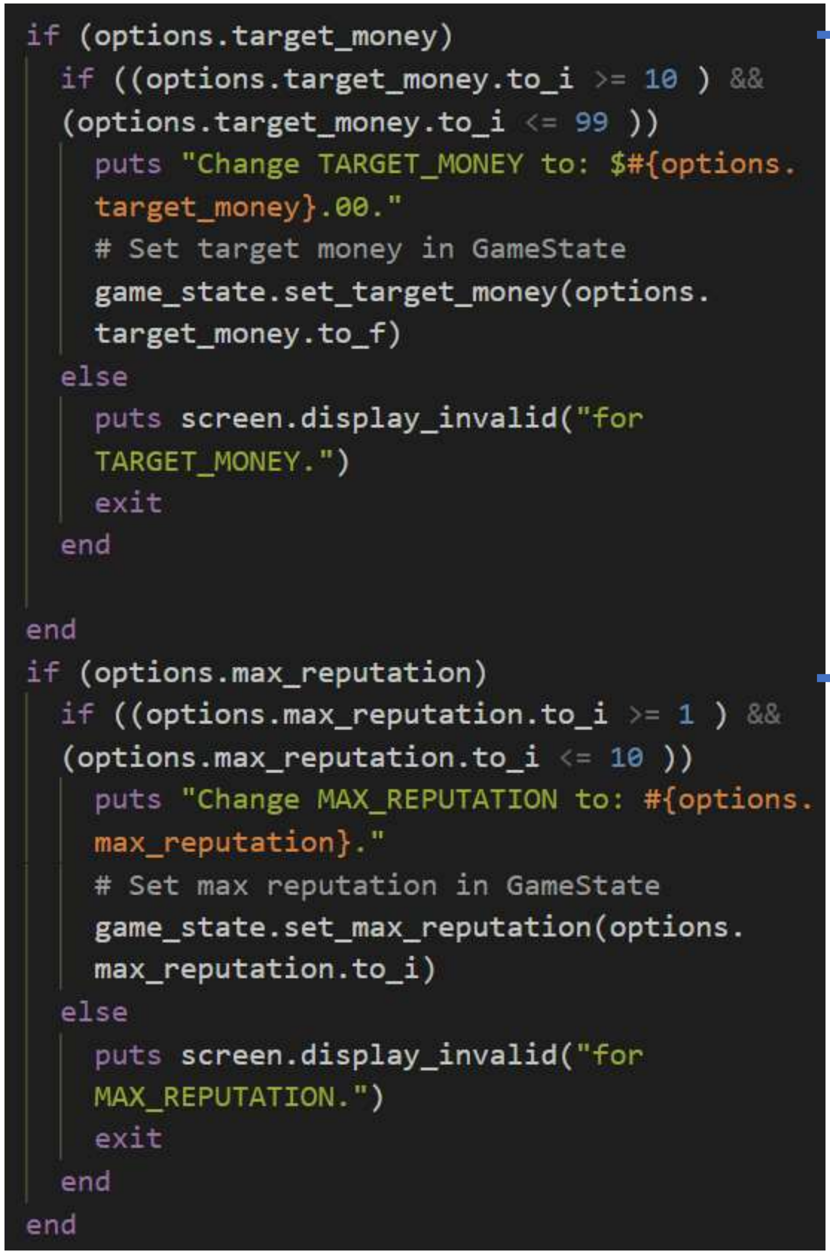
```
# Handle command line argument
opt_parser = OptionParser.new do |opt|
  opt.banner = "Usage (Gem's executable):
  start_burger_game [OPTION]\nOR\nUsage (bash
  script - install game): install.sh [OPTION]
  \nUsage (bash script - run game): burger_game.
  sh [OPTION]\n\n"

  opt.on("-h", "--help", "Print this Help menu
  for Burger Game.") do |arg|
    puts opt
    exit
  end

  opt.on("-m", "--money [TARGET_MONEY]", screen.
  display_h_money) { |arg| options.target_money
  = arg }

  opt.on("-r", "--reputation [MAX_REPUTATION]",
  screen.display_h_reputation) { |arg| options.
  max_reputation = arg }
end
```

Built-in argument parser

Collects value for option -m

Collects value for option -r

Command Line Argument(s) (cont.)



```ruby
if (options.target_money)
  if ((options.target_money.to_i >= 10 ) &&
  (options.target_money.to_i <= 99 ))
    puts "Change TARGET_MONEY to: $#{options.
    target_money}.00."
    # Set target money in GameState
    game_state.set_target_money(options.
    target_money.to_f)
  else
    puts screen.display_invalid("for
    TARGET_MONEY.")
    exit
  end

end
if (options.max_reputation)
  if ((options.max_reputation.to_i >= 1 ) &&
  (options.max_reputation.to_i <= 10 ))
    puts "Change MAX_REPUTATION to: #{options.
    max_reputation}."
    # Set max reputation in GameState
    game_state.set_max_reputation(options.
    max_reputation.to_i)
  else
    puts screen.display_invalid("for
    MAX_REPUTATION.")
    exit
  end
end
```

If user gives argument

Check if value within range

Display value

Call method to set custom value for target money

Value not valid, inform user, exit app

If user gives argument

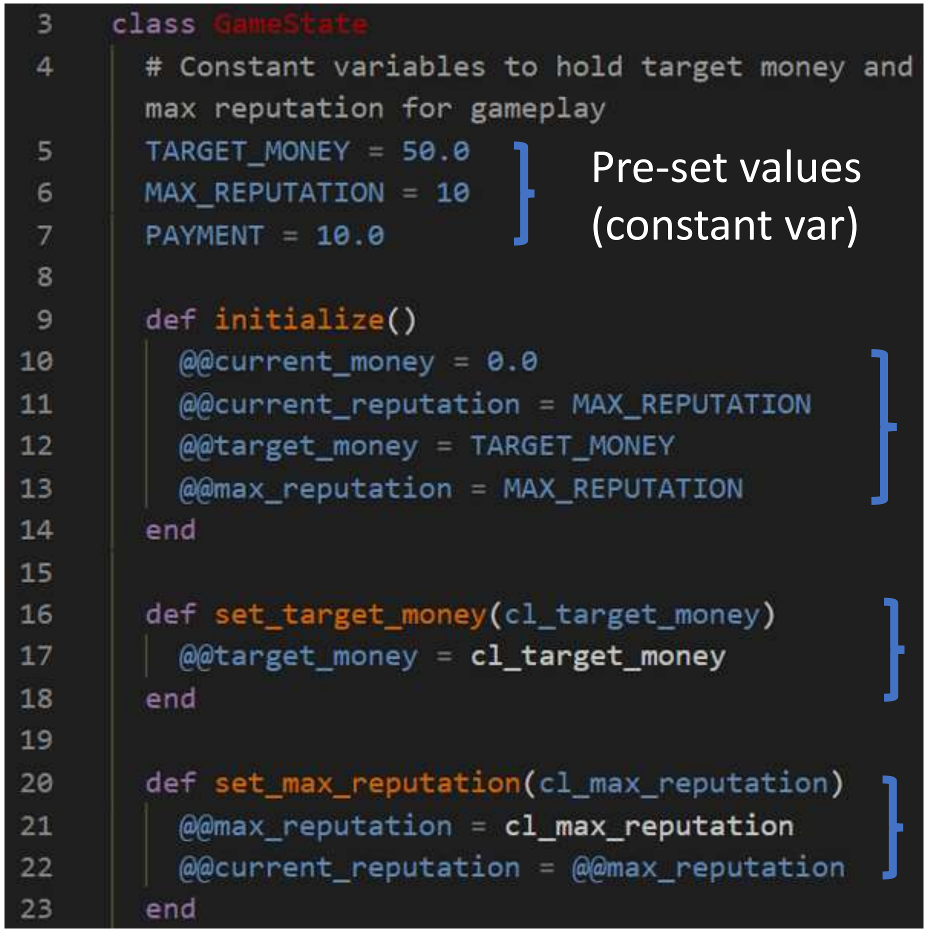
Check if value within range

Display value

Call method to set custom value for target money

Value not valid, inform user, exit app

GameState



```
class GameState
  # Constant variables to hold target money and
  max reputation for gameplay
  TARGET_MONEY = 50.0
  MAX_REPUTATION = 10
  PAYMENT = 10.0

  def initialize()
    @@current_money = 0.0
    @@current_reputation = MAX_REPUTATION
    @@target_money = TARGET_MONEY
    @@max_reputation = MAX_REPUTATION
  end

  def set_target_money(cl_target_money)
    @@target_money = cl_target_money
  end

  def set_max_reputation(cl_max_reputation)
    @@max_reputation = cl_max_reputation
    @@current_reputation = @@max_reputation
  end
```

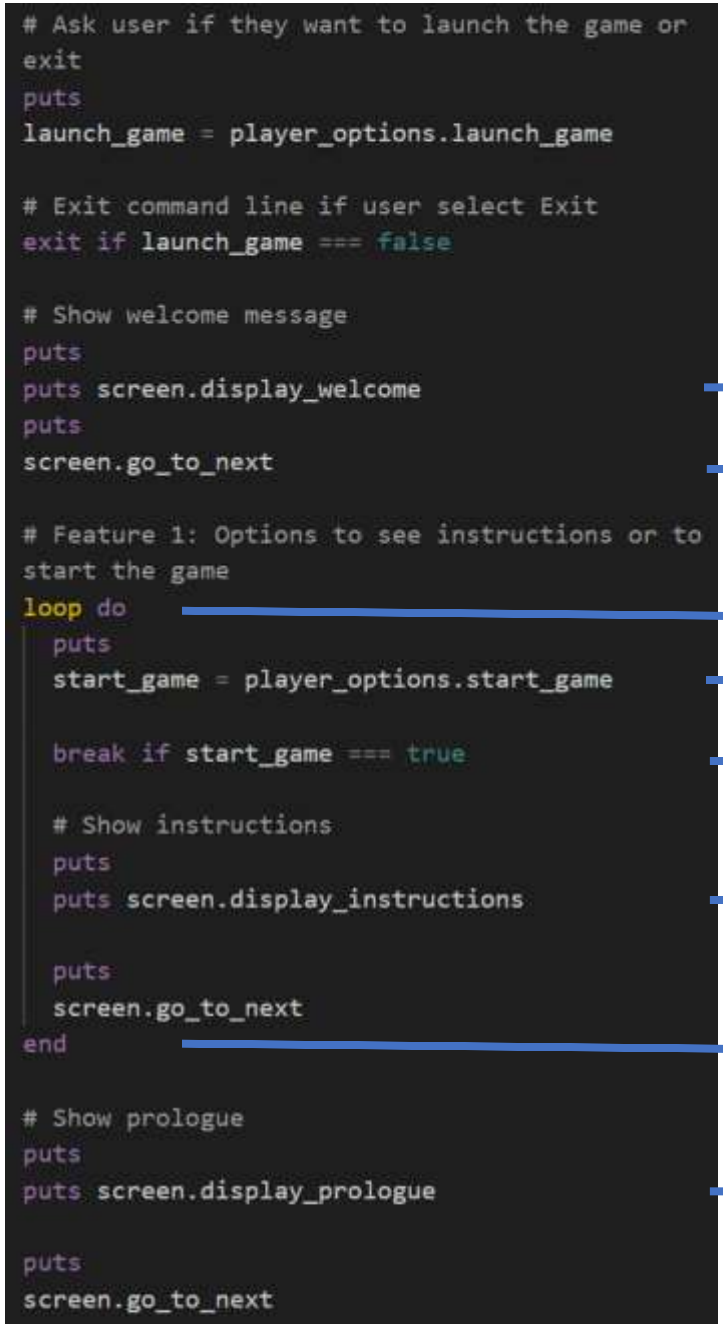
Pre-set values (constant var)

- Set current money to 0 and target money to pre-set value (constant var)
- Set current reputation and max reputation to pre-set value (constant var)

Receive a value (command line arg) and put into target money (replace pre-set value)

Receive a value (command line arg) and put into max reputation and current reputation (replace pre-set value)

Text Message Screen Output



```ruby
# Ask user if they want to launch the game or
exit
puts
launch_game = player_options.launch_game

# Exit command line if user select Exit
exit if launch_game === false

# Show welcome message
puts
puts screen.display_welcome
puts
screen.go_to_next

# Feature 1: Options to see instructions or to
start the game
loop do
  puts
  start_game = player_options.start_game

  break if start_game === true

  # Show instructions
  puts
  puts screen.display_instructions

  puts
  screen.go_to_next
end

# Show prologue
puts
puts screen.display_prologue

puts
screen.go_to_next
```

Display option to launch or exit game

Display formatted welcome message

Display options to go to next scene or to exit app

Option to view instructions or start game

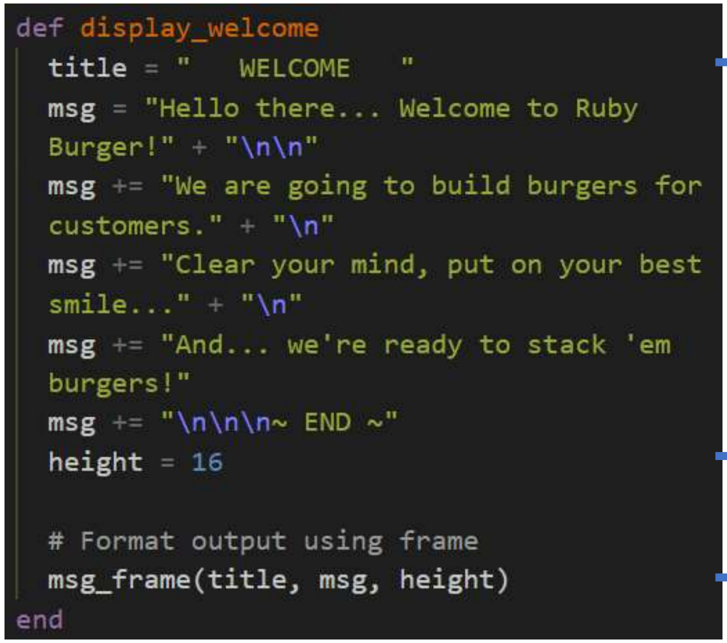
Break condition

Display instructions

Loop to display instructions until user choose start game (break condition)

Display prologue

Text Message Screen Output (cont.)



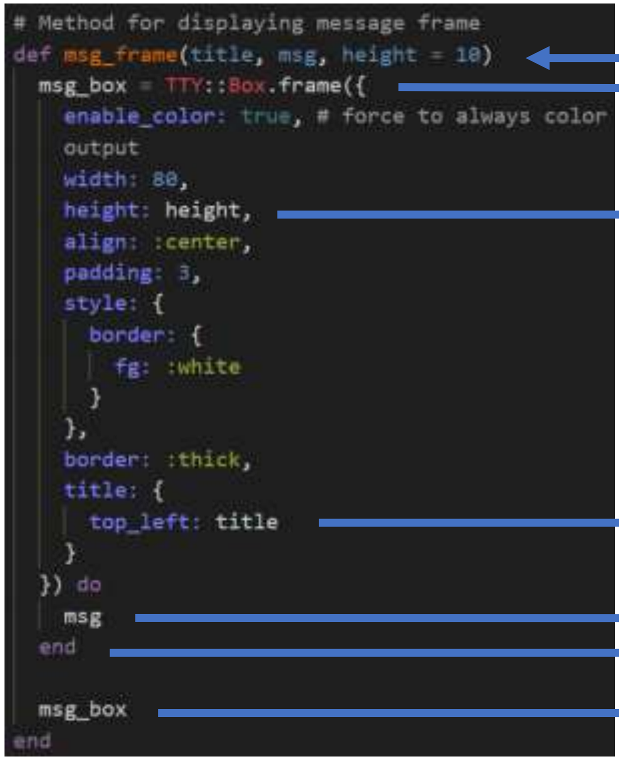
```ruby
def display_welcome
  title = "    WELCOME    "
  msg = "Hello there... Welcome to Ruby
  Burger!" + "\n\n"
  msg += "We are going to build burgers for
  customers." + "\n"
  msg += "Clear your mind, put on your best
  smile..." + "\n"
  msg += "And... we're ready to stack 'em
  burgers!"
  msg += "\n\n\n~ END ~"
  height = 16

  # Format output using frame
  msg_frame(title, msg, height)
end
```

Var to store title

Var store text message

Var to store frame height

Pass variables to method



```ruby
# Method for displaying message frame
def msg_frame(title, msg, height = 18)
  msg_box = TTY::Box.frame({
    enable_color: true, # force to always color
    output
    width: 80,
    height: height,
    align: :center,
    padding: 3,
    style: {
      border: {
        fg: :white
      }
    },
    border: :thick,
    title: {
      top_left: title
    }
  }) do
    msg
  end

  msg_box
end
```

Frame height

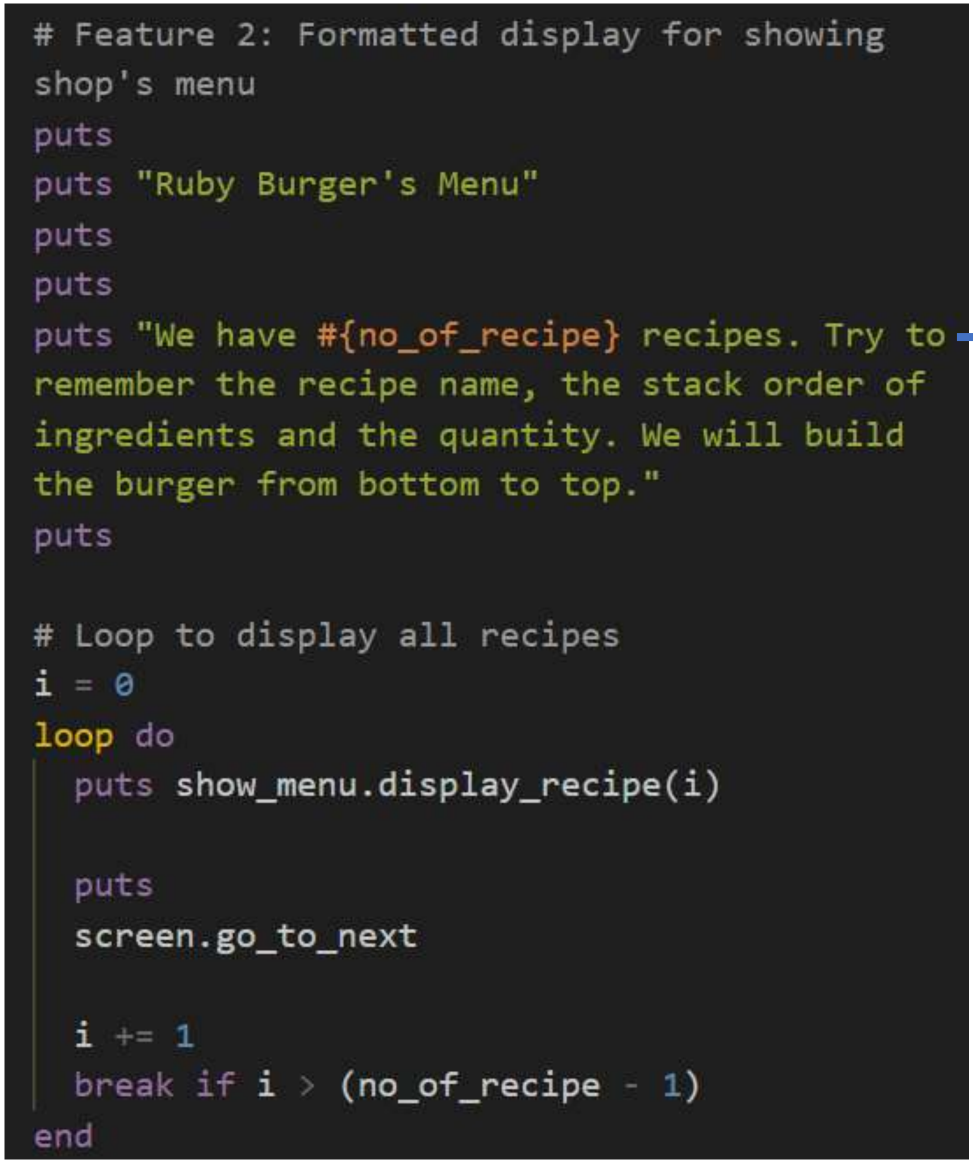
Method provided by TTY Box

Title

Text message

Return to be displayed

Loop to Display Shop's Menu (Recipes)

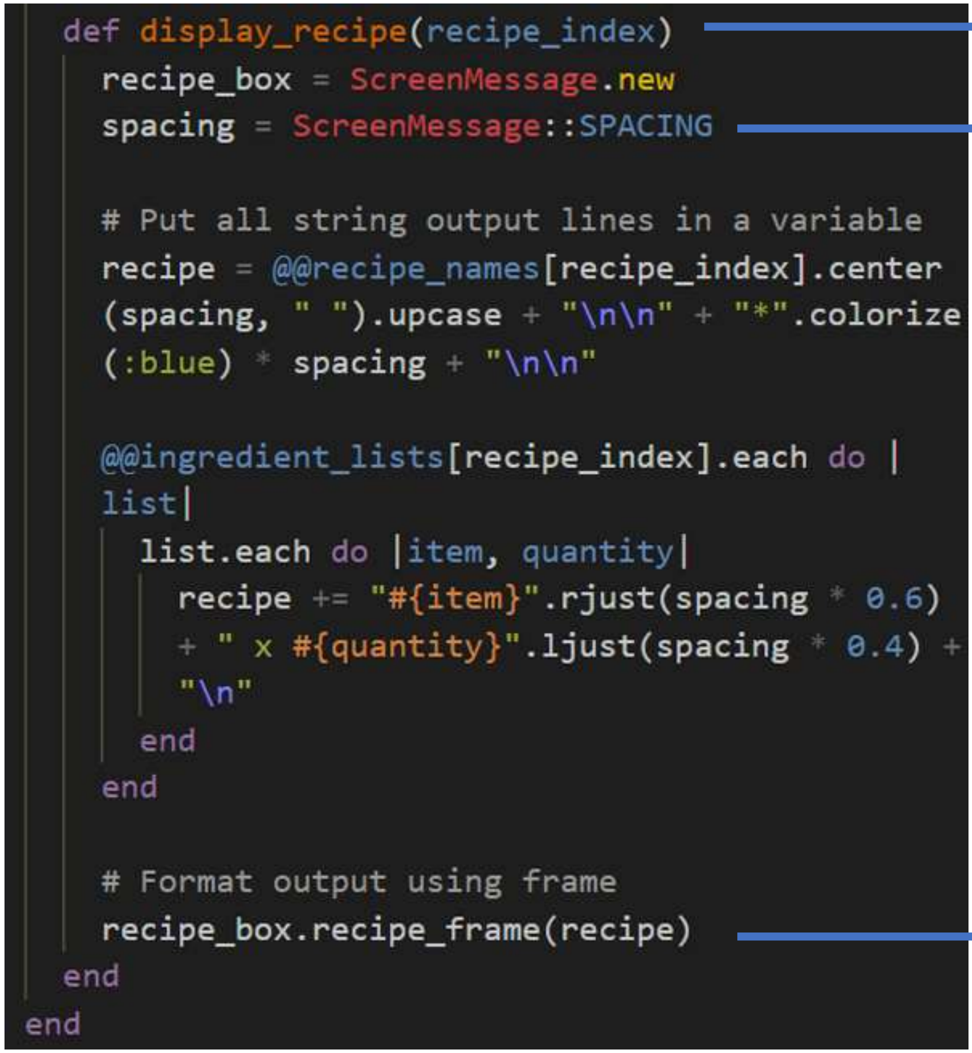


```ruby
# Feature 2: Formatted display for showing
shop's menu
puts
puts "Ruby Burger's Menu"
puts
puts
puts "We have #{no_of_recipe} recipes. Try to
remember the recipe name, the stack order of
ingredients and the quantity. We will build
the burger from bottom to top."
puts

# Loop to display all recipes
i = 0
loop do
  puts show_menu.display_recipe(i)

  puts
  screen.go_to_next

  i += 1
  break if i > (no_of_recipe - 1)
end
```

Print total number of recipe available in JSON file.

Loop through all available recipes and print to screen

Loop to Display Shop's Menu (Recipes) (cont.)



```
  def display_recipe(recipe_index)
    recipe_box = ScreenMessage.new
    spacing = ScreenMessage::SPACING

    # Put all string output lines in a variable
    recipe = @@recipe_names[recipe_index].center
    (spacing, " ").upcase + "\n\n" + "*".colorize
    (:blue) * spacing + "\n\n"

    @@ingredient_lists[recipe_index].each do |
    list|
      list.each do |item, quantity|
        recipe += "#{item}".rjust(spacing * 0.6)
        + " x #{quantity}".ljust(spacing * 0.4) +
        "\n"
      end
    end

    # Format output using frame
    recipe_box.recipe_frame(recipe)
  end
end
```

Receive value *i*, use as index no. to get recipe data at *i* index in array

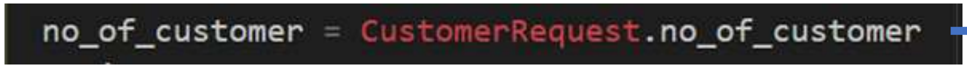
Just grabbing value for formatting purpose

Put recipe name and separator in var

Loop through array of arrays of hashes to put ingredient-quantity pairs to the var

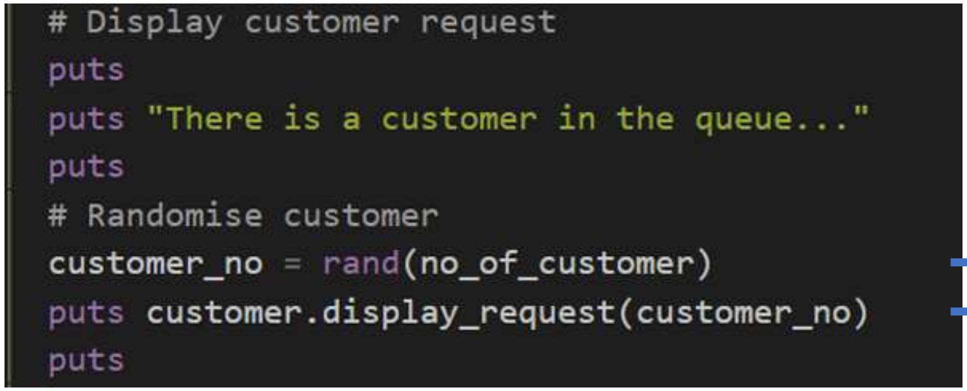
Pass the var to method that formats the frame for print output

Randomised Customer Request



```ruby
no_of_customer = CustomerRequest.no_of_customer
```

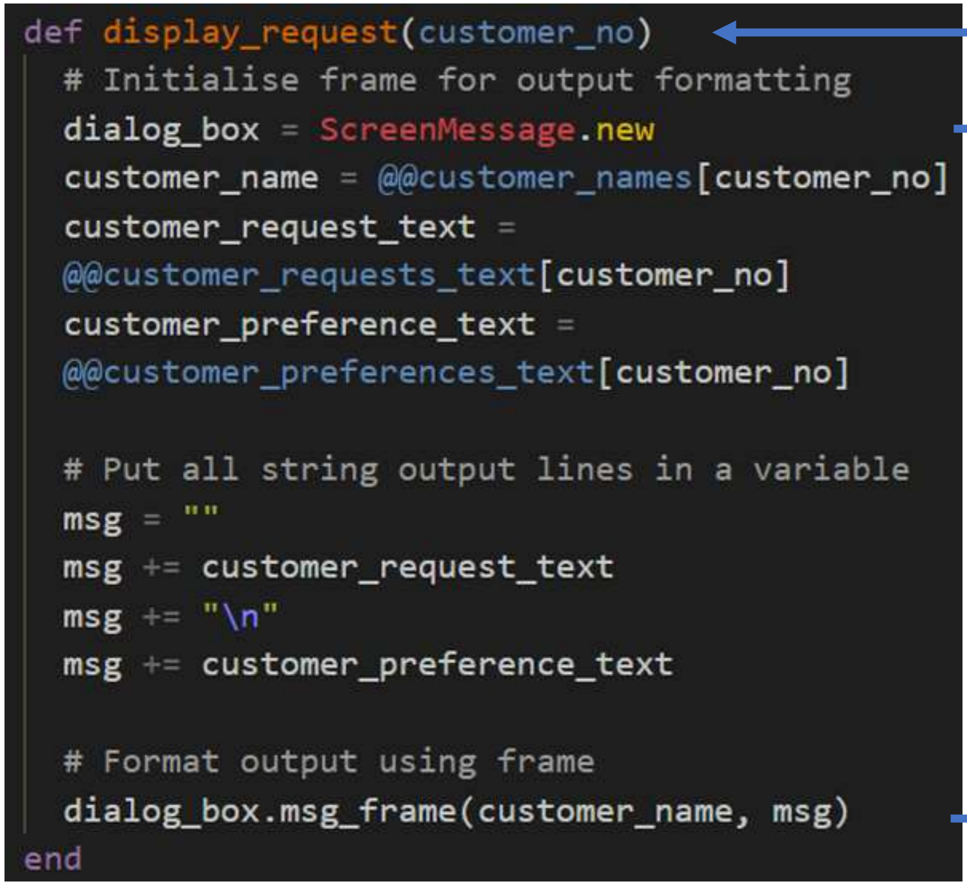
Get number of customers available in JSON file



```ruby
# Display customer request
puts
puts "There is a customer in the queue..."
puts
# Randomise customer
customer_no = rand(no_of_customer)
puts customer.display_request(customer_no)
puts
```

Randomise using *rand()*

Pass the randomised number to method



```ruby
def display_request(customer_no)
  # Initialise frame for output formatting
  dialog_box = ScreenMessage.new
  customer_name = @@customer_names[customer_no]
  customer_request_text =
  @@customer_requests_text[customer_no]
  customer_preference_text =
  @@customer_preferences_text[customer_no]

  # Put all string output lines in a variable
  msg = ""
  msg += customer_request_text
  msg += "\n"
  msg += customer_preference_text

  # Format output using frame
  dialog_box.msg_frame(customer_name, msg)
end
```

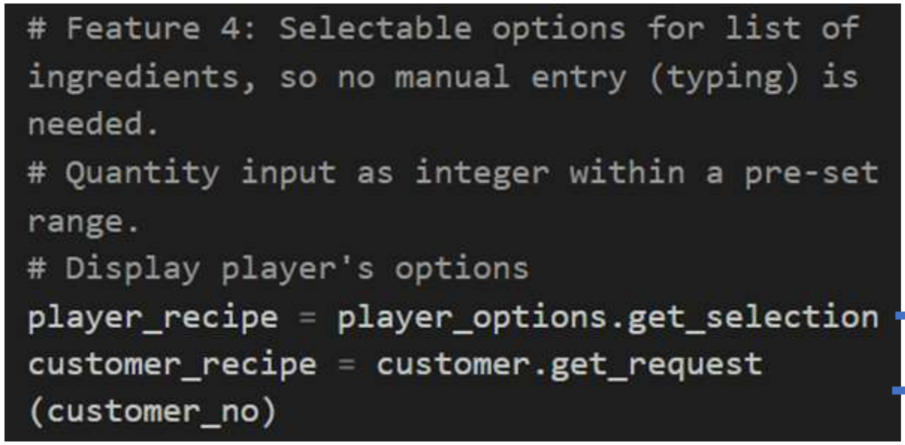
Create instance

Put required values into variables

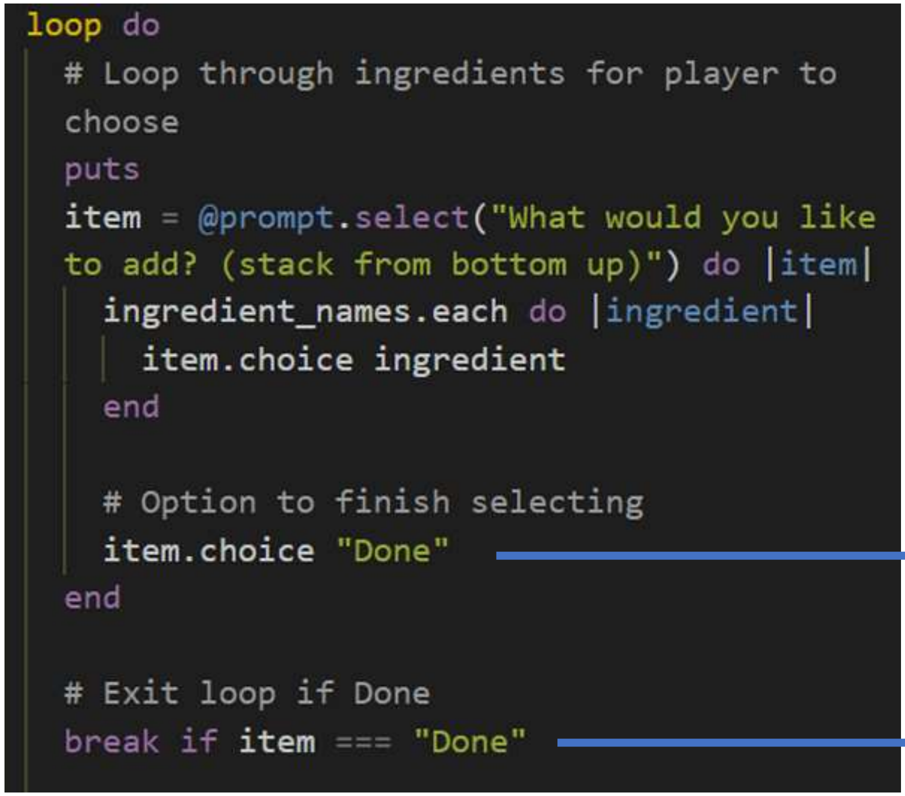
Put text message into variable

Pass the var to method that formats the frame for print output

Selectable options and validated text input



```
# Feature 4: Selectable options for list of
ingredients, so no manual entry (typing) is
needed.
# Quantity input as integer within a pre-set
range.
# Display player's options
player_recipe = player_options.get_selection
customer_recipe = customer.get_request
(customer_no)
```



```
loop do
  # Loop through ingredients for player to
  choose
  puts
  item = @prompt.select("What would you like
  to add? (stack from bottom up)") do |item|
    ingredient_names.each do |ingredient|
      item.choice ingredient
    end

    # Option to finish selecting
    item.choice "Done"
  end

  # Exit loop if Done
  break if item === "Done"
```

Display options and store input in variable

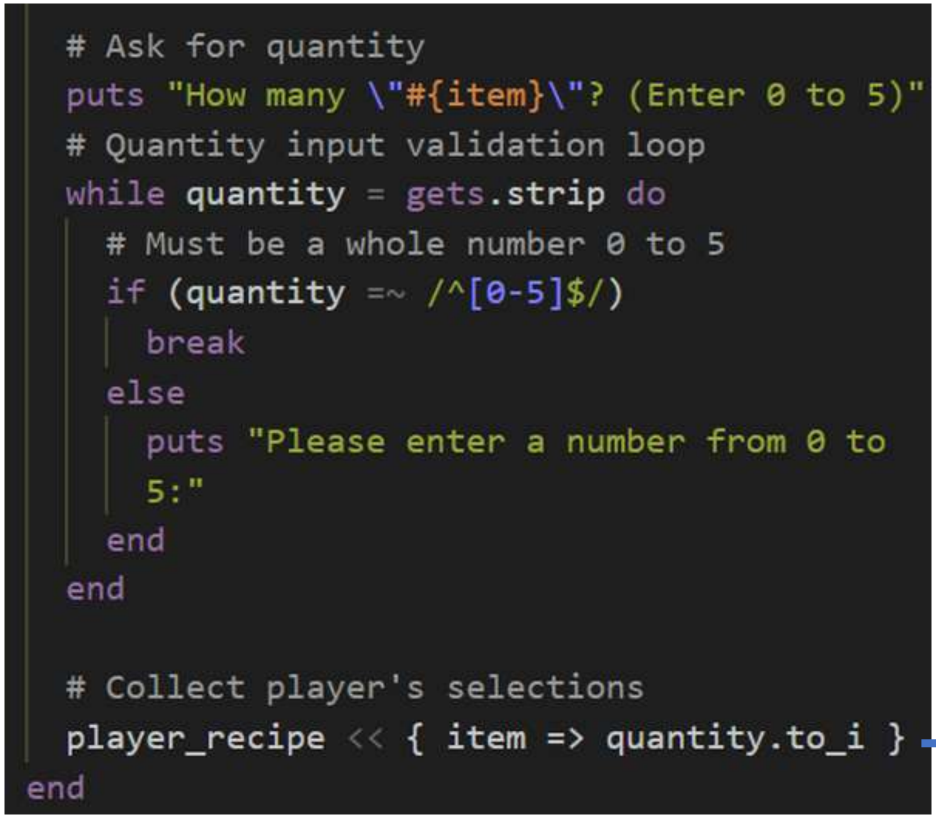
Create array of ingredient-quantity as per customer request

Loop to display all available ingredients (TTY Prompt gem), store input in variable

Additional option

Break condition

Selectable options and validated text input (cont.)

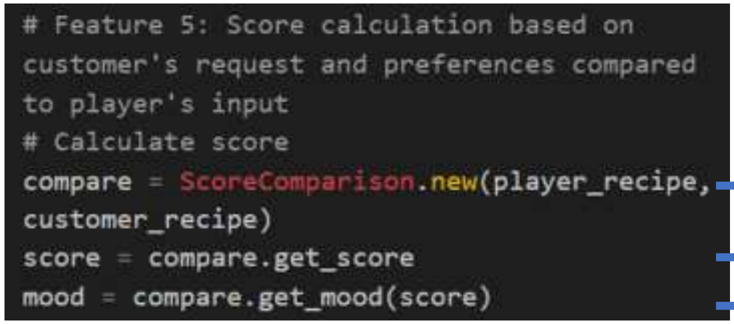


```
  # Ask for quantity
  puts "How many \"#{item}\"? (Enter 0 to 5)"
  # Quantity input validation loop
  while quantity = gets.strip do
    # Must be a whole number 0 to 5
    if (quantity =~ /^[0-5]$/)
      break
    else
      puts "Please enter a number from 0 to
      5:"
    end
  end

  # Collect player's selections
  player_recipe << { item => quantity.to_i }
end
```

Input validation (Int 0-5)

Pair input ingredient-quantity, push to variable

Score Calculations

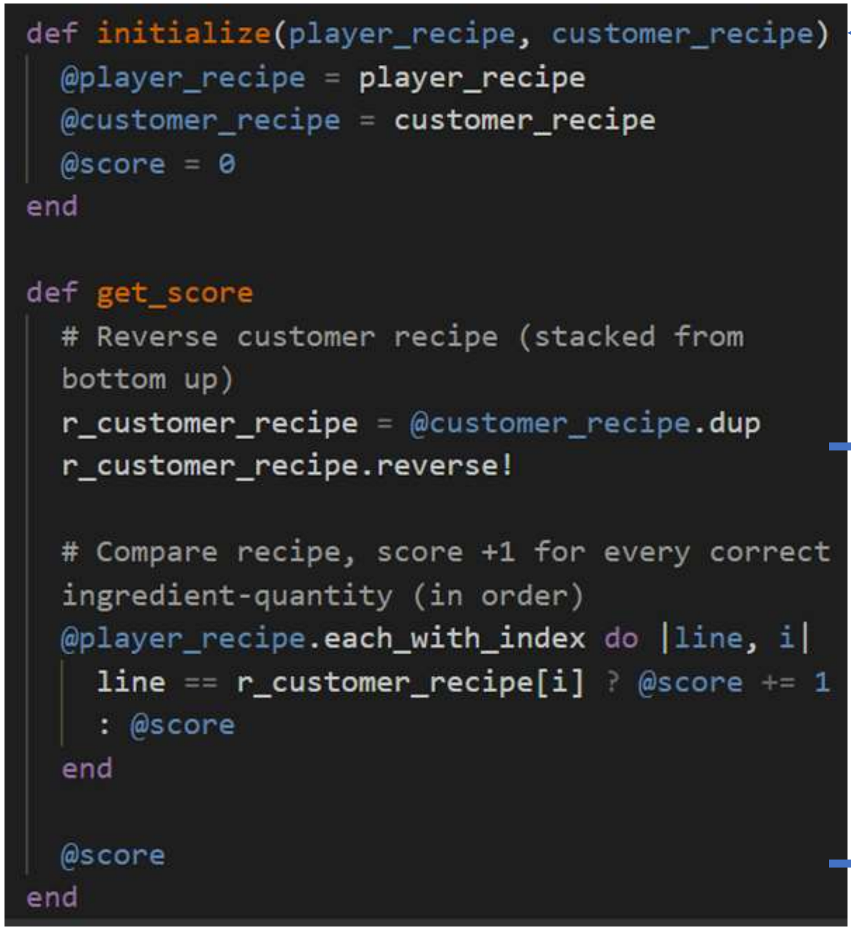


```
# Feature 5: Score calculation based on
customer's request and preferences compared
to player's input
# Calculate score
compare = ScoreComparison.new(player_recipe,
customer_recipe)
score = compare.get_score
mood = compare.get_mood(score)
```

Create instance, pass values for comparison (score calculation)

Get score value

Get customer's mood value by passing the score value



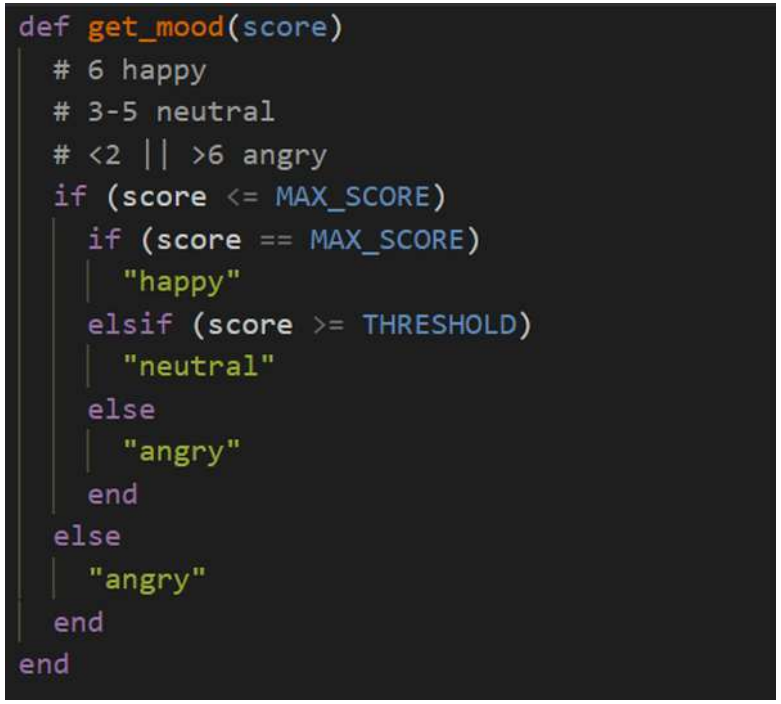
```
def initialize(player_recipe, customer_recipe)
  @player_recipe = player_recipe
  @customer_recipe = customer_recipe
  @score = 0
end

def get_score
  # Reverse customer recipe (stacked from
  bottom up)
  r_customer_recipe = @customer_recipe.dup
  r_customer_recipe.reverse!

  # Compare recipe, score +1 for every correct
  ingredient-quantity (in order)
  @player_recipe.each_with_index do |line, i|
    line == r_customer_recipe[i] ? @score += 1
    : @score
  end

  @score
end
```

Create duplicate array, reverse (stack bottom to top)

Compare user input with customer's request,
Add +1 to score for every matching line

Return score

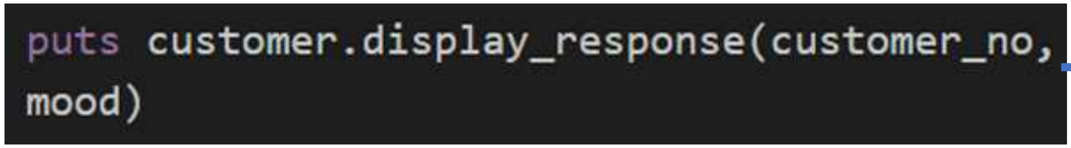
Score Calculations (cont.)



```
def get_mood(score)
  # 6 happy
  # 3-5 neutral
  # <2 || >6 angry
  if (score <= MAX_SCORE)
    if (score == MAX_SCORE)
      "happy"
    elsif (score >= THRESHOLD)
      "neutral"
    else
      "angry"
    end
  else
    "angry"
  end
end
```

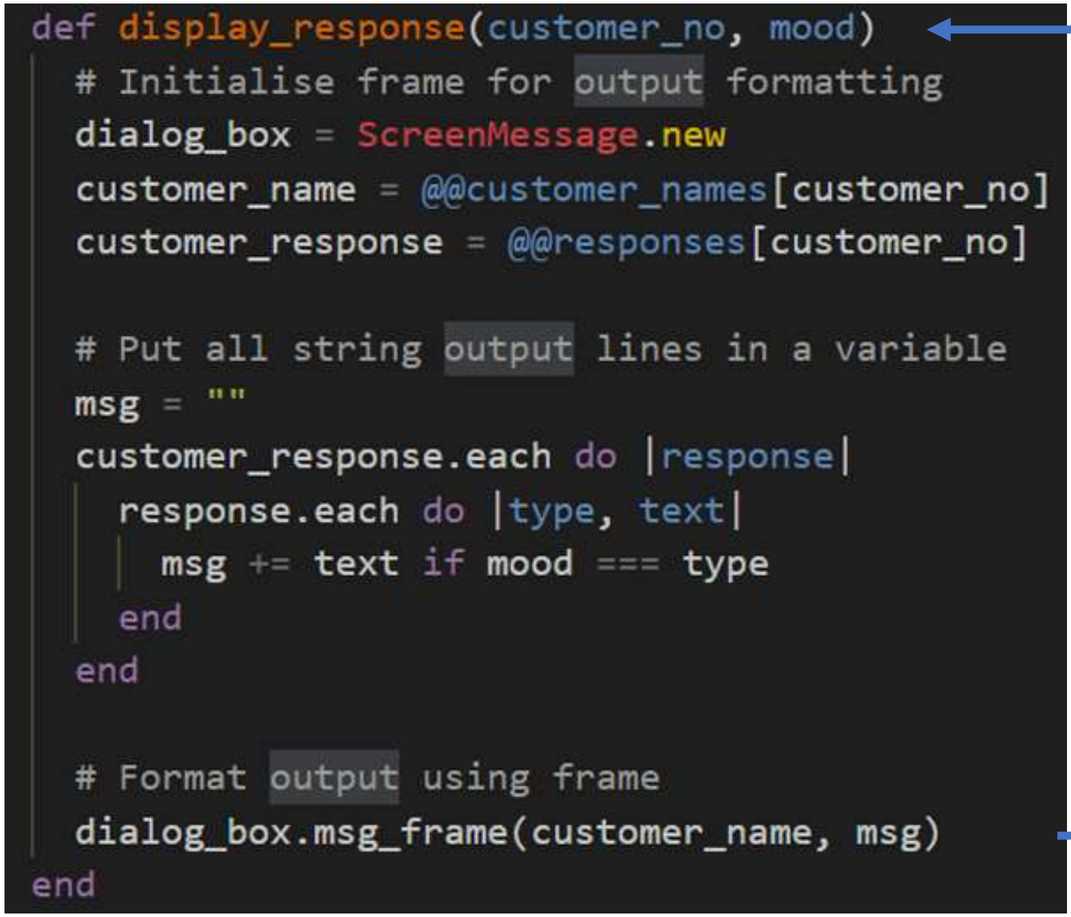
Conditional control structure
Returns “happy”/”neutral”/”angry” based on score value passed to it

Display Correct Customer's Response



```ruby
puts customer.display_response(customer_no,
mood)
```

Pass values to method



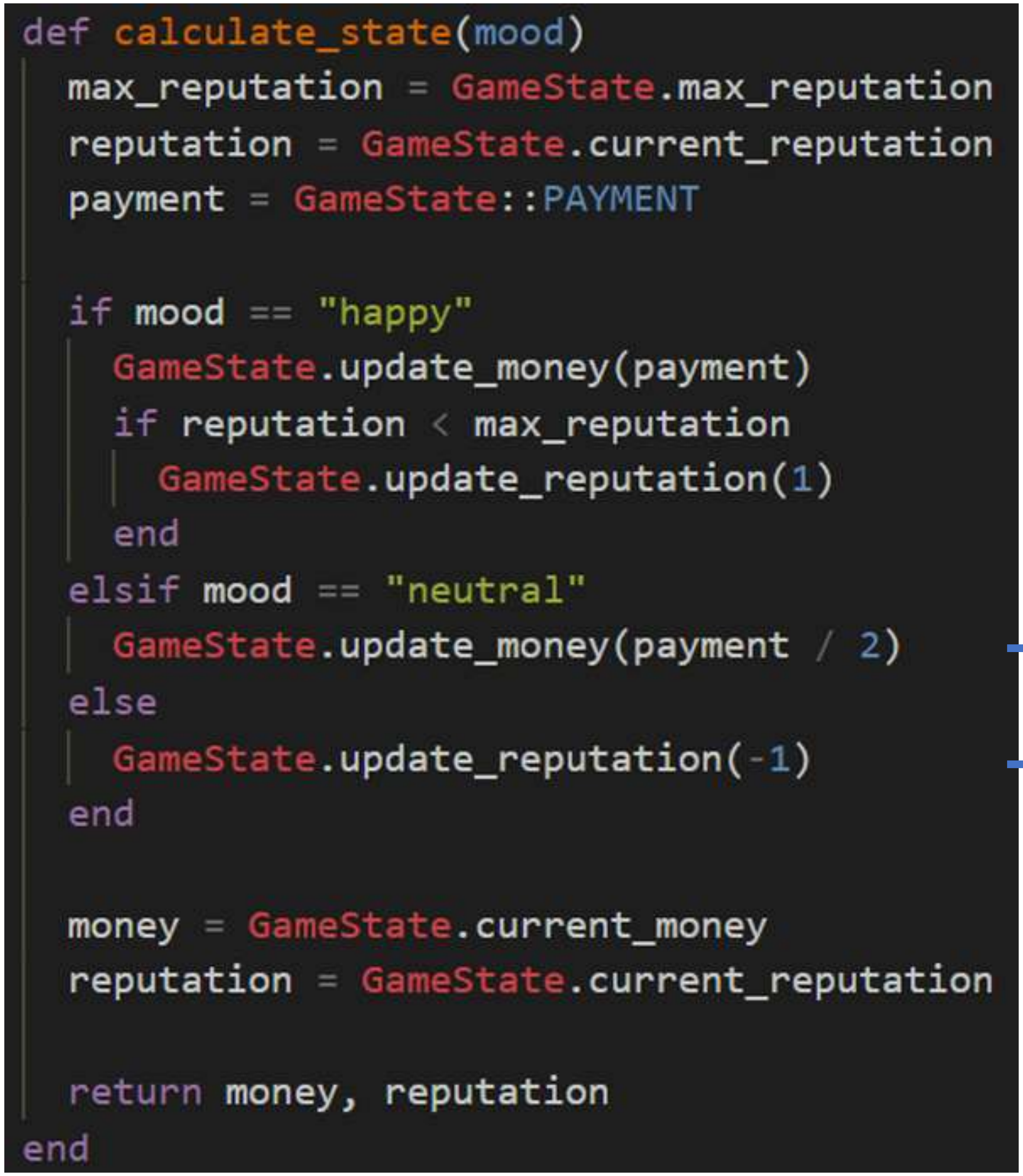
```ruby
def display_response(customer_no, mood)
  # Initialise frame for output formatting
  dialog_box = ScreenMessage.new
  customer_name = @@customer_names[customer_no]
  customer_response = @@responses[customer_no]

  # Put all string output lines in a variable
  msg = ""
  customer_response.each do |response|
    response.each do |type, text|
      msg += text if mood === type
    end
  end

  # Format output using frame
  dialog_box.msg_frame(customer_name, msg)
end
```

Put required values in variables

Loop through array of arrays of hashes

Pass the var to method that formats the frame for print output

Score Calculations (cont.)



```
def calculate_state(mood)
  max_reputation = GameState.max_reputation
  reputation = GameState.current_reputation
  payment = GameState::PAYMENT

  if mood == "happy"
    GameState.update_money(payment)
    if reputation < max_reputation
      GameState.update_reputation(1)
    end
  elsif mood == "neutral"
    GameState.update_money(payment / 2)
  else
    GameState.update_reputation(-1)
  end

  money = GameState.current_money
  reputation = GameState.current_reputation

  return money, reputation
end
```

Collect values from GameState

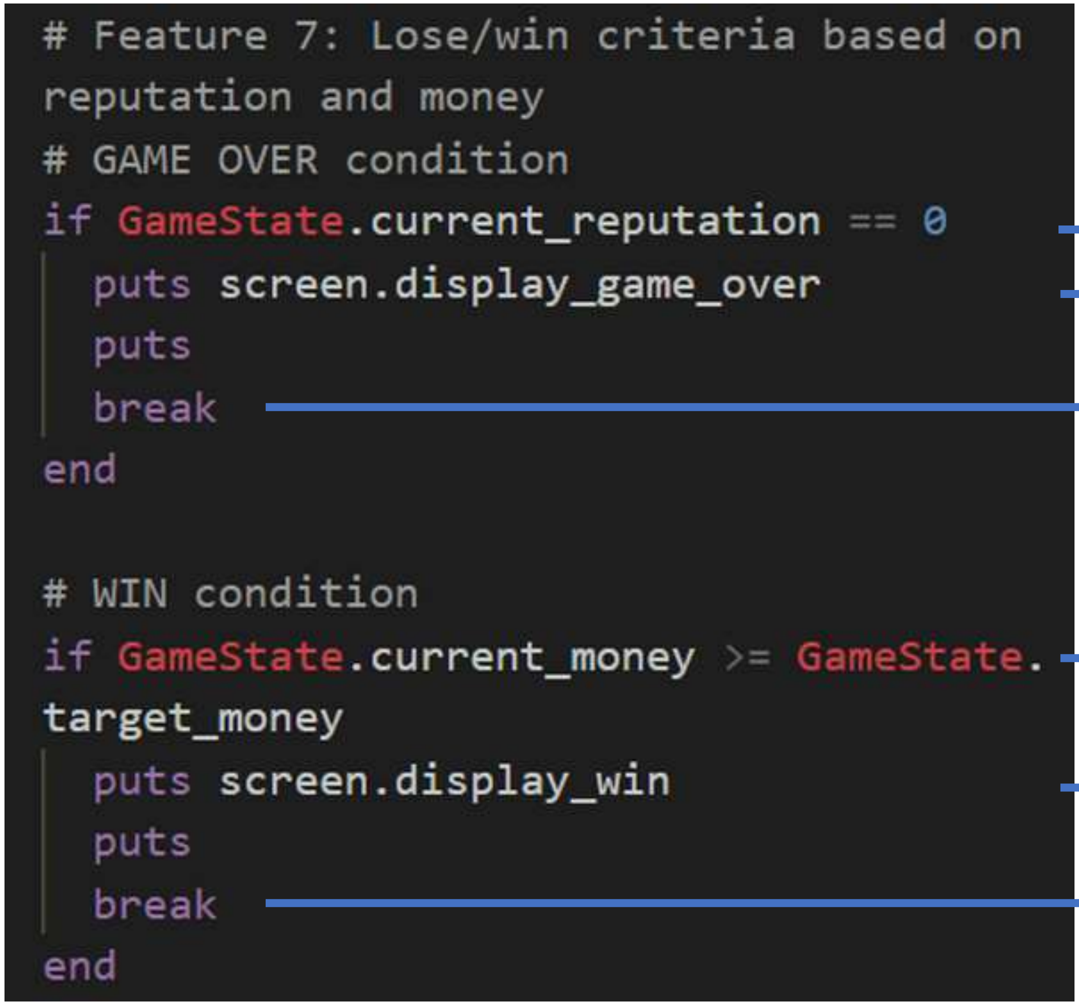
Add money and reputation (only if reputation is below max)

Add half amount of money

Subtract reputation

Return updated money and reputation for test case purpose

Win and Game Over Conditions



```
# Feature 7: Lose/win criteria based on
reputation and money
# GAME OVER condition
if GameState.current_reputation == 0
  puts screen.display_game_over
  puts
  break
end

# WIN condition
if GameState.current_money >= GameState.
target_money
  puts screen.display_win
  puts
  break
end
```

Game Over condition

Display Game Over message

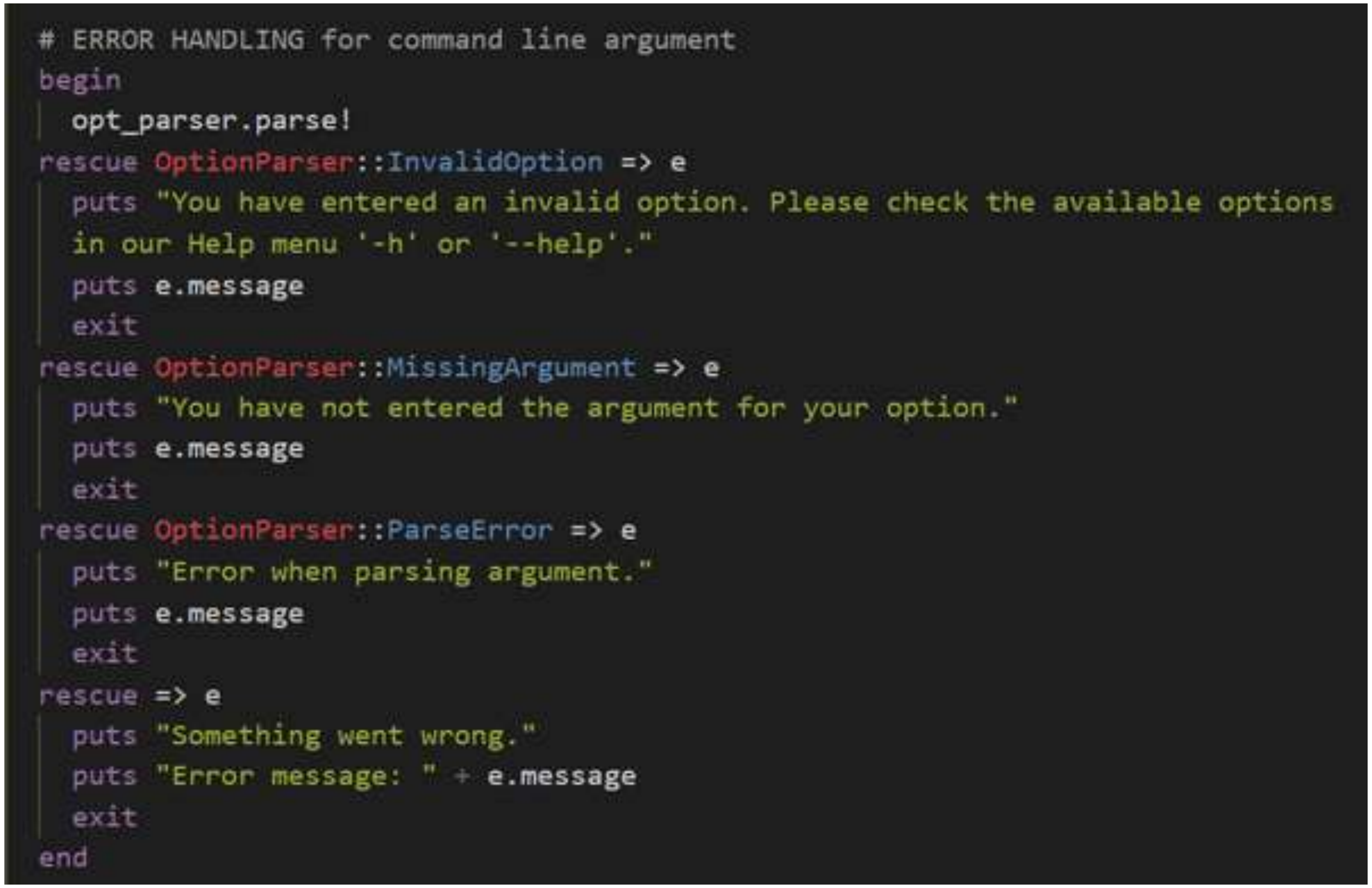
Break out of the game loop, exit the app

Win condition

Display Win message

Break out of the game loop, exit the app

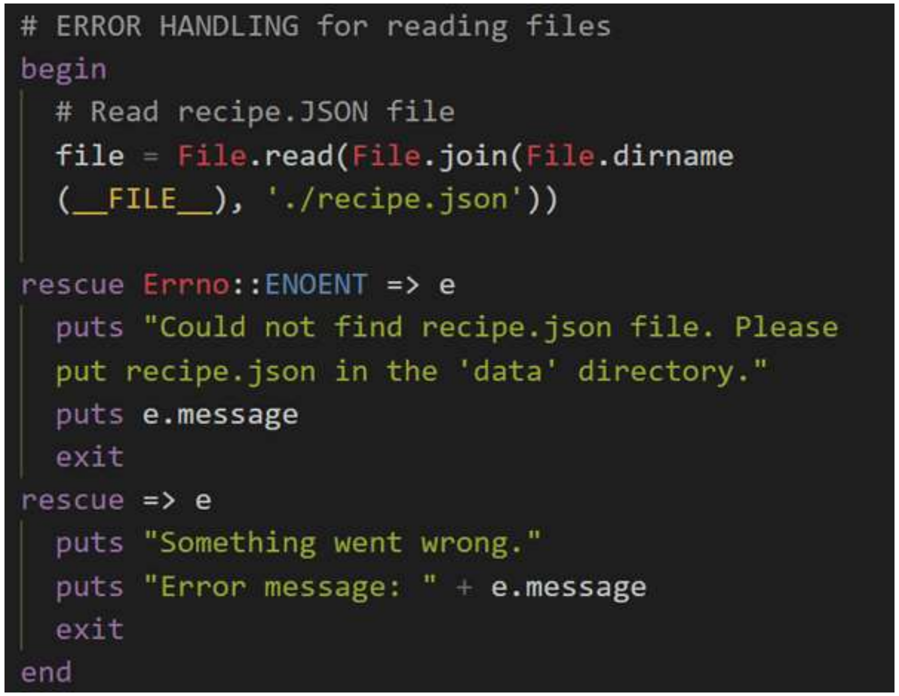
Error Handling



```ruby
# ERROR HANDLING for command line argument
begin
  opt_parser.parse!
rescue OptionParser::InvalidOption => e
  puts "You have entered an invalid option. Please check the available options
  in our Help menu '-h' or '--help'."
  puts e.message
  exit
rescue OptionParser::MissingArgument => e
  puts "You have not entered the argument for your option."
  puts e.message
  exit
rescue OptionParser::ParseError => e
  puts "Error when parsing argument."
  puts e.message
  exit
rescue => e
  puts "Something went wrong."
  puts "Error message: " + e.message
  exit
end
```

- Handles argument parsing error for invalid option, missing arguments, parsing error, and other standard errors.

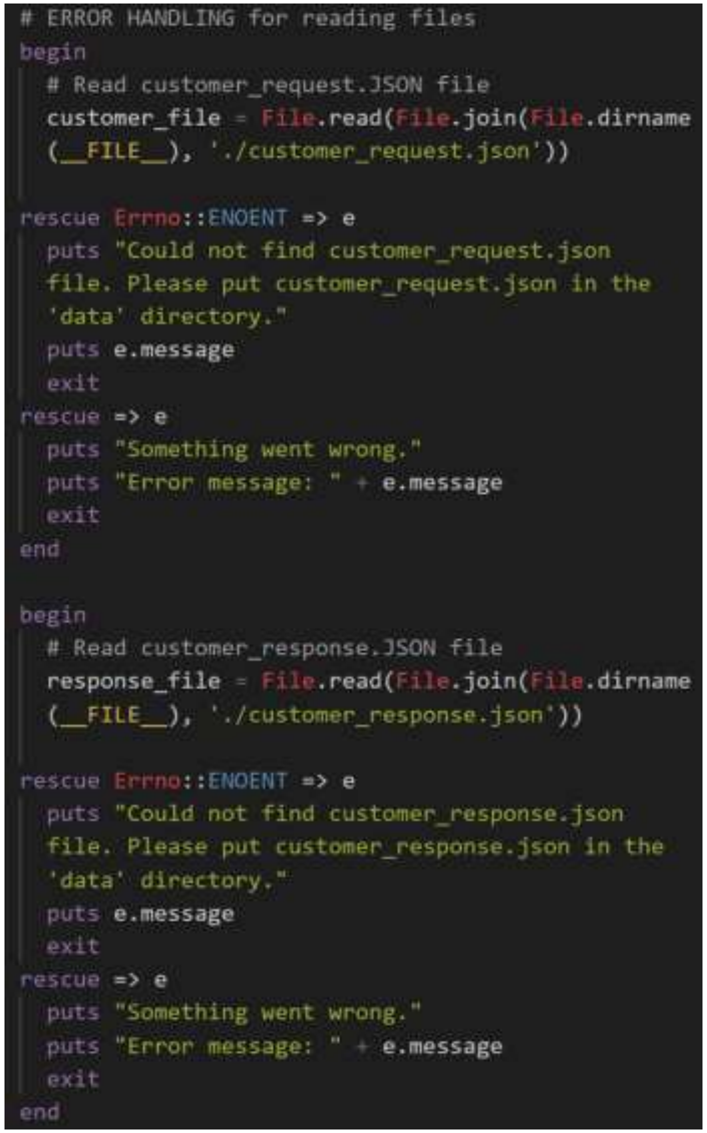
Error Handling (cont.)



```
# ERROR HANDLING for reading files
begin
  # Read recipe.JSON file
  file = File.read(File.join(File.dirname
  (__FILE__), './recipe.json'))

rescue Errno::ENOENT => e
  puts "Could not find recipe.json file. Please
  put recipe.json in the 'data' directory."
  puts e.message
  exit
rescue => e
  puts "Something went wrong."
  puts "Error message: " + e.message
  exit
end
```

- Handles error for file not found and other standard errors.

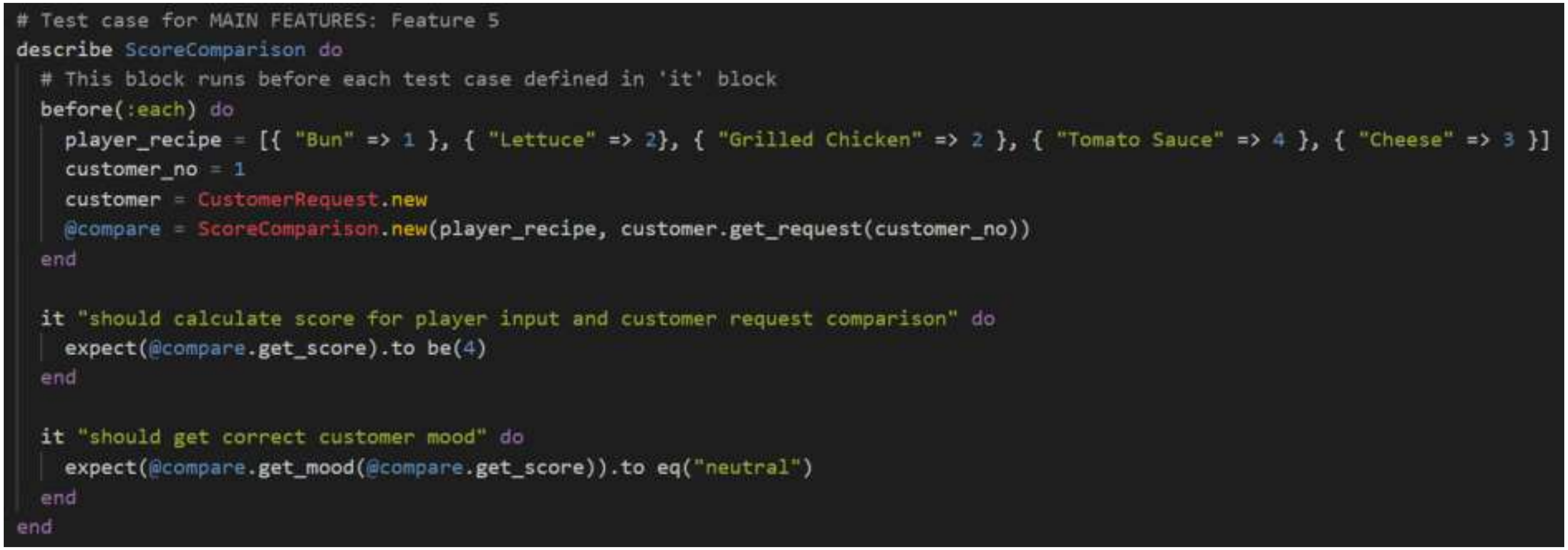
Error Handling (cont.)



```
# ERROR HANDLING for reading files
begin
  # Read customer_request.JSON file
  customer_file = File.read(File.join(File.dirname
  (__FILE__), './customer_request.json'))

rescue Errno::ENOENT => e
  puts "Could not find customer_request.json
  file. Please put customer_request.json in the
  'data' directory."
  puts e.message
  exit
rescue => e
  puts "Something went wrong."
  puts "Error message: " + e.message
  exit
end

begin
  # Read customer_response.JSON file
  response_file = File.read(File.join(File.dirname
  (__FILE__), './customer_response.json'))

rescue Errno::ENOENT => e
  puts "Could not find customer_response.json
  file. Please put customer_response.json in the
  'data' directory."
  puts e.message
  exit
rescue => e
  puts "Something went wrong."
  puts "Error message: " + e.message
  exit
end
```

- Handles error for file not found and other standard errors.

Test Cases

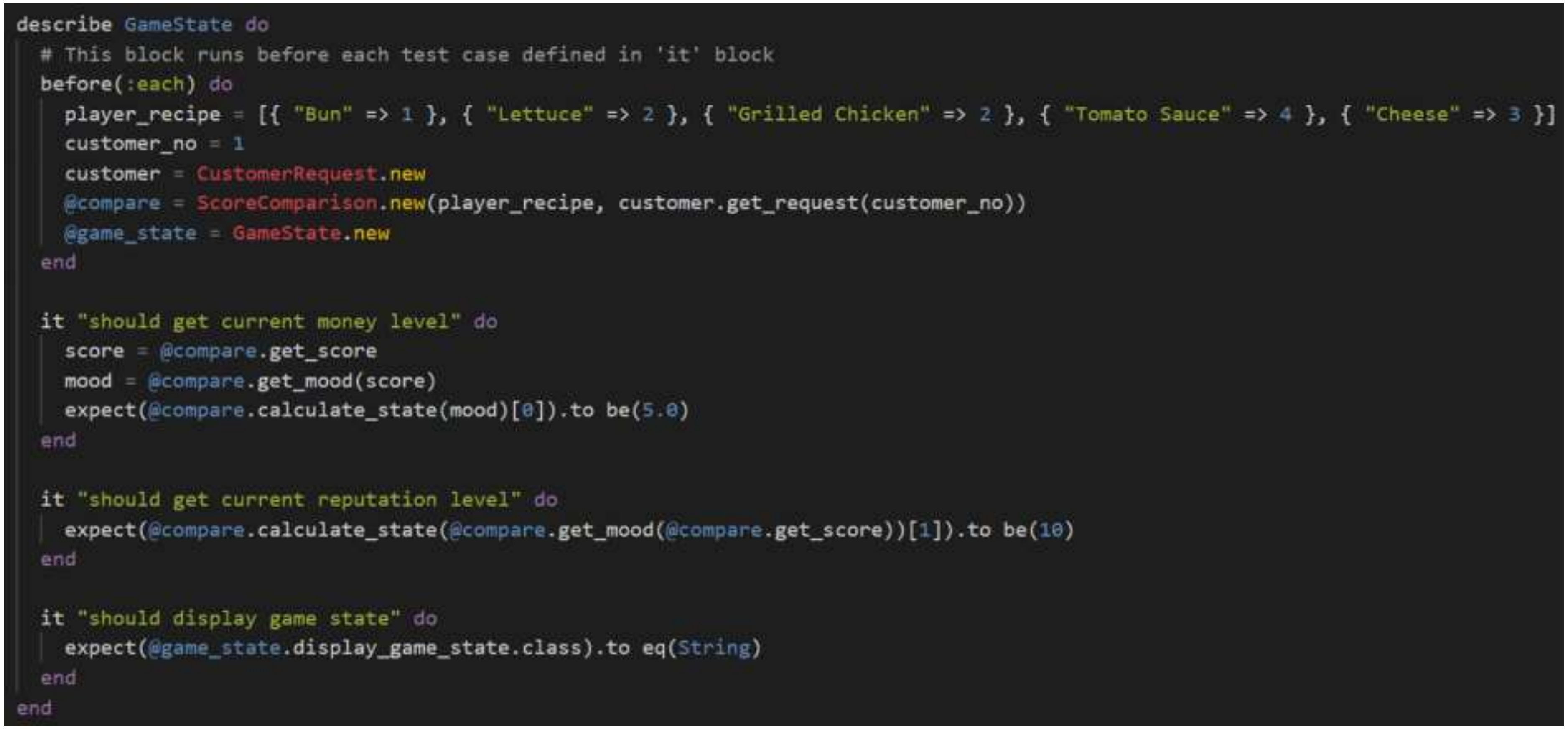


```
# Test case for MAIN FEATURES: Feature 5
describe ScoreComparison do
  # This block runs before each test case defined in 'it' block
  before(:each) do
    player_recipe = [{ "Bun" => 1 }, { "Lettuce" => 2}, { "Grilled Chicken" => 2 }, { "Tomato Sauce" => 4 }, { "Cheese" => 3 }]
    customer_no = 1
    customer = CustomerRequest.new
    @compare = ScoreComparison.new(player_recipe, customer.get_request(customer_no))
  end

  it "should calculate score for player input and customer request comparison" do
    expect(@compare.get_score).to be(4)
  end

  it "should get correct customer mood" do
    expect(@compare.get_mood(@compare.get_score)).to eq("neutral")
  end
end
```

Some of test cases used to check:

- Gives correct total score?
- Gives correct mood type?

Test Cases (cont.)



```
describe GameState do
  # This block runs before each test case defined in 'it' block
  before(:each) do
    player_recipe = [{ "Bun" => 1 }, { "Lettuce" => 2 }, { "Grilled Chicken" => 2 }, { "Tomato Sauce" => 4 }, { "Cheese" => 3 }]
    customer_no = 1
    customer = CustomerRequest.new
    @compare = ScoreComparison.new(player_recipe, customer.get_request(customer_no))
    @game_state = GameState.new
  end

  it "should get current money level" do
    score = @compare.get_score
    mood = @compare.get_mood(score)
    expect(@compare.calculate_state(mood)[0]).to be(5.0)
  end

  it "should get current reputation level" do
    expect(@compare.calculate_state(@compare.get_mood(@compare.get_score))[1]).to be(10)
  end

  it "should display game state" do
    expect(@game_state.display_game_state.class).to eq(String)
  end
end
```

Some of test cases used to check:

- Gives correct current money amount?
- Gives correct current reputation points?
- Gives output?

4. Review

Challenges, Ethical Issues, Favourite Part

Challenges

- Inexperience in estimating **scale** and **scope** of project
  - Not enough time to create good **content**
  - Not enough time to build better **UI** and **UX**
  - Not enough time to document everything properly
- Unresolved issue/bug (array handling)
  - Re-did parts of code in CustomerRequest and Recipe classes

Ethical Issues

- ASCII Art
  - Image of a burger and chips created with ASCII characters included in the app (in Win message) is owned by Joan G. Stark and attribution is given as per directed by the artist.

- Ruby Gems
  - TTY Prompt, TTY Box, Artii: MIT Licence
  - Colorize: GPL-2.0 Licence
  - JSON: Ruby Licence

Favourite Part

- Project planning and documentations as practice for **common standard procedure** used in professional projects.
- Experience using project **planning tools** (e.g., diagram, project management tool).
- Practice **Object-Oriented Design** application.
- Practice **Test Driven Development** approach, experience using testing tool (Rspec).

Thank You