



T1A3 - Terminal App

Brayden O'Gorman



FLIP BET

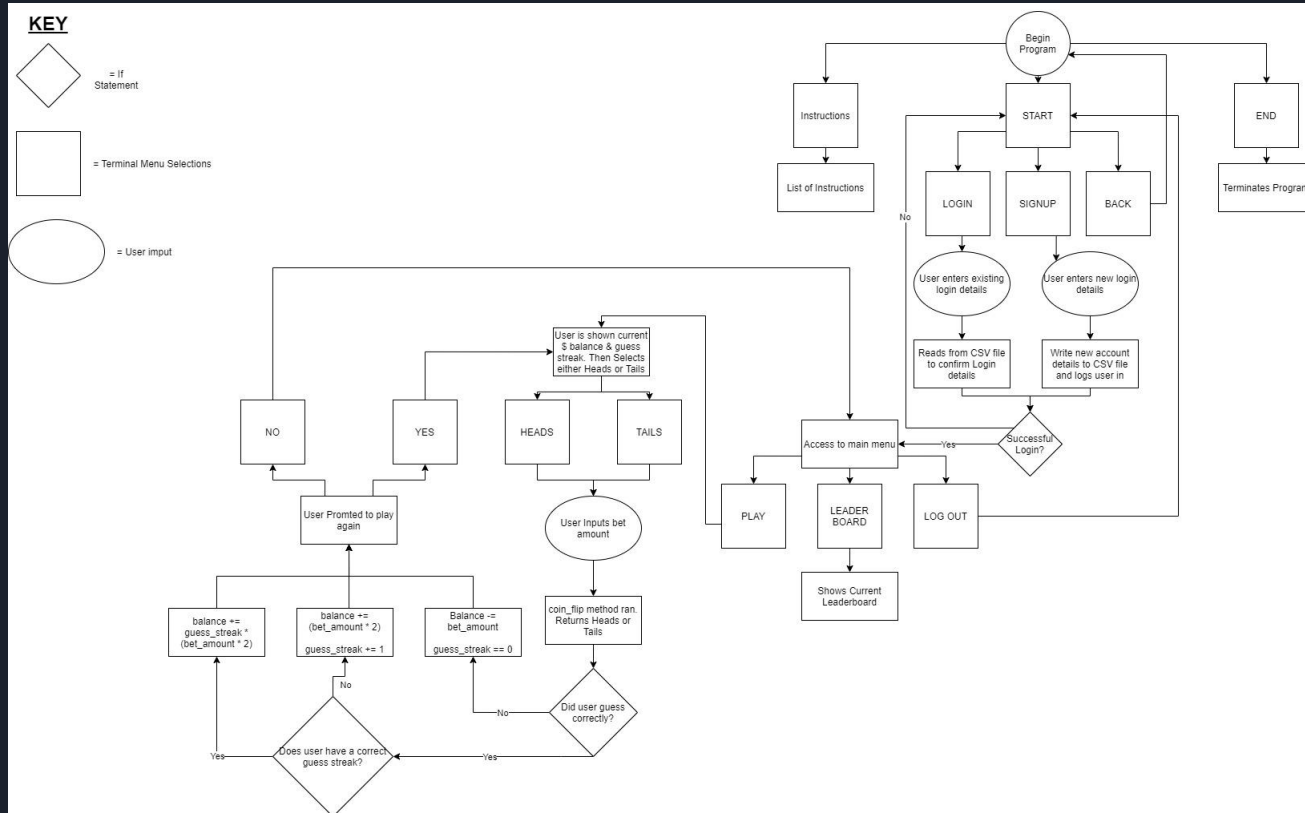
Flip bet is a terminal based app that allows users to make bets on whether a coin will flip heads or tails.

Each user begins at \$250 after signing up for an account. Users will be able to log in and out of their account and their balance and guess streak will be saved.

Users will be rewarded with a multiplier bonus for consecutive guesses.

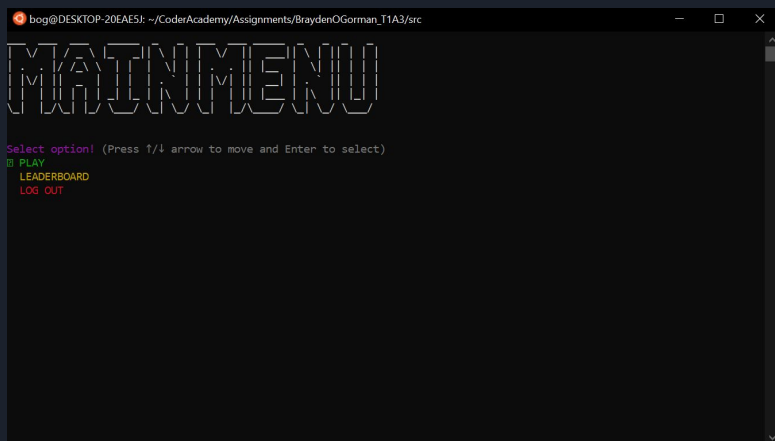
The goal? To beat the current record holders Jeff Bezos, Elon Musk and Bill Gates, The leaderboard can be accessed via the main menu once logged in.

CONTROL FLOW



Features

TTY - Prompt/TTY - Font: I used the TTY - Prompt/ TTY - Font ruby gems to create an easy to use terminal menu that allows the user to navigate the menu by using the arrow keys and Enter key. TTY - Font will also provide a cleaner UI. Examples shows below



```
bog@DESKTOP-20EAE5: ~/CoderAcademy/Assignments/BraydenOGorman_T1A3/src
MAINMENU

Select option! (Press ↑/↓ arrow to move and Enter to select)
❏ PLAY
  LEADERBOARD
  LOG OUT
```



```
bog@DESKTOP-20EAE5: ~/CoderAcademy/Assignments/BraydenOGorman_T1A3/src
Heads or Tails?

Select whether you think the coin will land on Heads or Tails?
❏ HEADS
  TAILS
  BACK
```

Features

Play: The Play feature allows user to bet their user \$ balance on whether a coin will flip on heads or tails. Once the user selects the play feature in the main menu it will run the play method which will give the user the ability to select Heads or Tails (This selection is run from the coin_flip_selection method). Then input the amount they would like to bet (This selection is run from the user_bet_amount method). A reward of double the bet amount will be given to the user if they guess correctly. The play method will also check to see if the user is currently on a guess streak. If they are they will be rewarded with a bet multiplier. The user balance and current guess streak are accessed by a global variable \$current_user[]. Once the result has been output. The users will be prompted if they would like to play again.

```
def play
  balance = $current_user[:balance].to_i
  streak = $current_user[:streak].to_i
  font = font_instance
  if coin_flip == coin_flip_selection
    system("clear")
    $current_user[:streak] = streak + 1
    if streak >= 2
      $current_user[:balance] = balance + (user_bet_amount * streak)
      system("clear")
      puts "YOU WON!!!".colorize(:green) + " New Balance: ${balance} ||" + " Current Win Streak Multiplier: #{streak}x".colorize(:blue)
    else
      $current_user[:balance] = balance + user_bet_amount
      system("clear")
      puts "YOU WON!!!".colorize(:green) + " New Balance: ${balance} ||" + " Current Win Streak: #{streak}".colorize(:blue)
    end
  else
    $current_user[:balance] = balance - user_bet_amount
    $current_user[:streak] = streak - streak
    system("clear")
    puts "oof, sorry that was incorrect. NEW BALANCE: ${balance} || Current Win Streak: #{streak}".colorize(:red)
    if balance == 0
      system("clear")
      $current_user[:balance] = balance + 50
      puts "I can see you ran out of money. Here is $50 on the house. Goodluck! NEW BALANCE: ${balance}"
    end
  end
end
play_again
end
```

Features

Sign up/Login: Upon starting the app. Users will be prompted to either sign up or log in. If users select Sign up, they will be asked to enter their user details. The username and password along side a default \$ balance and streak will be added to their account and stored in a CSV file. If a user is trying to log in. They will be prompted to enter their username/password and it will be checked against the CSV file to confirm the log in and the current user details are stored in a hash assigned to the `$current_user` variable.

```
def sign_up
  prompt = prompt_instance
  balance = 250
  streak = 0
  username, password = get_user_details
  username_taken = find_username(username)
  if username_taken == false
    $users.push([username, password, balance, streak])
  else
    system('clear')
    prompt.select("The username you have entered already exists. Please try again") do |menu|
      menu.choice "TRY AGAIN", -> {sign_up}
      menu.choice "BACK TO MENU".colorize(:red), ->{start_menu}
    end
  end
end

login_menu
end
```

```
def login
  prompt = prompt_instance
  username, password = get_user_details
  user_data = find_username(username)
  if user_data
    if user_data[1] == password
      $current_user[:username] = user_data[0]
      $current_user[:password] = user_data[1]
      $current_user[:balance] = user_data[2]
      $current_user[:streak] = user_data[3]
      puts "Hello #{$current_user[:username]}! Your current balance is: $#{ $current_user[:balance]} "
      prompt.select("Successful Login!", show_help: :never) do |menu|
        menu.choice "CONTINUE", -> {main_menu}
      end
    else
      prompt.select("Incorrect Username or Password. Please try again") do |menu|
        menu.choice "TRY AGAIN", -> {login}
        menu.choice "BACK TO MENU".colorize(:red), -> {start_menu}
      end
    end
  else
    prompt.select("Incorrect Username or Password. Please try again") do |menu|
      menu.choice "TRY AGAIN", -> {login}
      menu.choice "BACK TO MENU".colorize(:red), -> {start_menu}
    end
  end
end
```

Features

Leaderboard: When the leaderboard method is run, it will update the current balance and streak of the user stored in the \$users variable then writing that to the CSV. The current users will then be pushed to a hash and sorted in order of their balance to be displayed to the user.

```
# I want to take each array in user data, output to hash and order them by :balance value to be displayed to the user
def leaderboard
  update_user_data
  font = font_instance
  prompt = prompt_instance
  system('clear')
  puts font.write("LEADERBOARD")
  puts "-----"
  scores = []
  $users.each do |x|
    scores << {username: x[0], score: x[2].to_i}
  end
  scores.sort_by! { |hash| hash[:score] }
  scores.reverse!
  scores.each do |x|
    puts "#{x[:username]} SCORE: #{x[:score]}".colorize(:magenta)
  end
  prompt.select("", show_help: :never) do |menu|
    menu.choice "Back".colorize(:red), -> {main_menu}
  end
end
```

LEADERBOARD

Jeff Bezos	SCORE: \$100000
Elon Musk	SCORE: \$75000
Bill Gates	SCORE: \$50000
brayydz	SCORE: \$850

► Back



Development Process

- Started planning process by identifying and writing out the main features of my application
- Once I knew what my features would be, I then searched for Ruby Gems to use
- Use of Trello allowed me to recognize my MVP features and layout development plan.
- Creating a control flow diagram helped me better understand the overall build process and code structure.
- Constant use of Trello throughout the coding process to keep me on track.



Challenges/Ethical Issues/Favourite Parts

Challenges:

- Actually implementing the skills that I have learnt and practiced.
- Learning how to interact with CSV/JSON files

Ethical Issues:

- No ethical issues. List of Ruby Gems included are in the README

Favourite Parts:

- Seeing everything come together and working.
- Realising that I have learnt quite a lot throughout Term 1 and was able to use the skills that I have learned to create something.



Thank you!