Smesh Bets

Connor Roberts

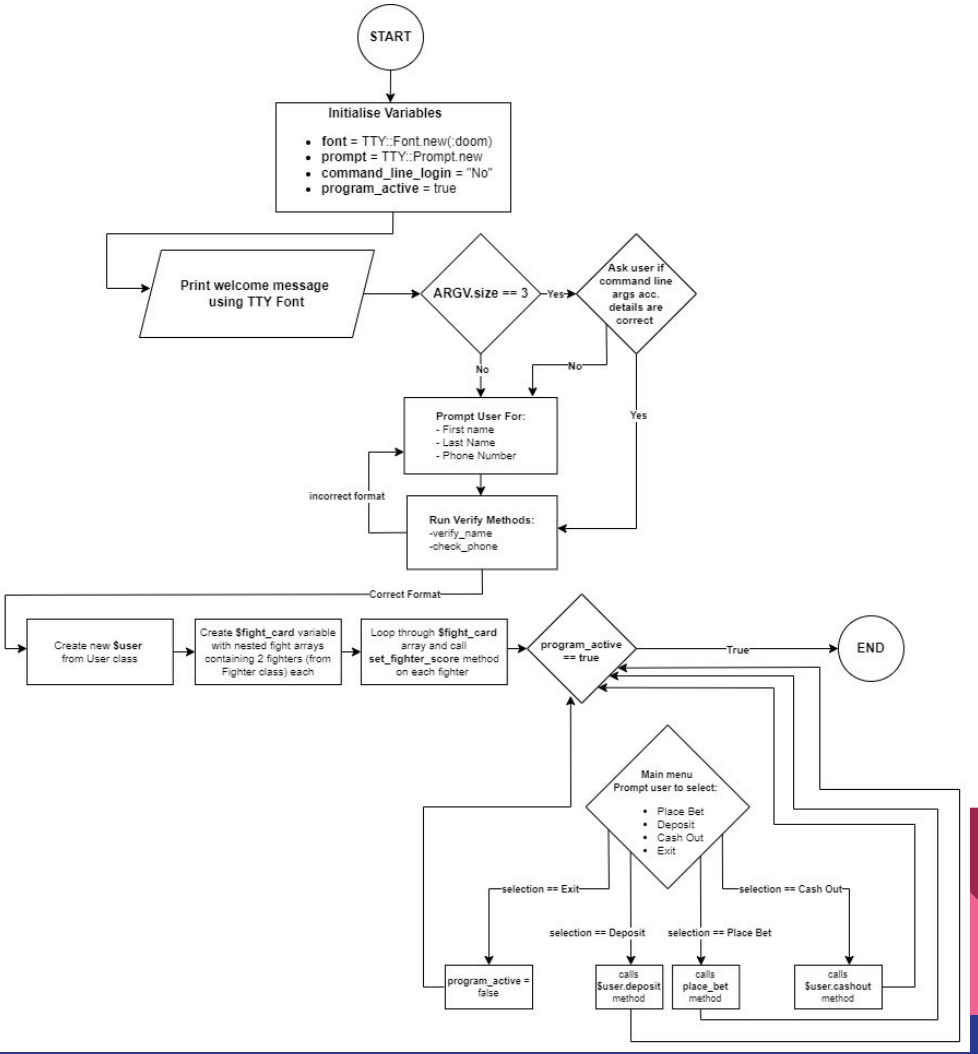
Application Synopsis

Smesh Bets is a Mixed Martial Arts betting platform built in the Ruby terminal, which allows users to view and bet on upcoming contests.

Features

1. User can deposit funds into their account
2. User can place bets on upcoming fights & win/lose money based on the outcome
3. User can withdraw funds from their account

Smesh Bets index.rb
App Logic Flowchart

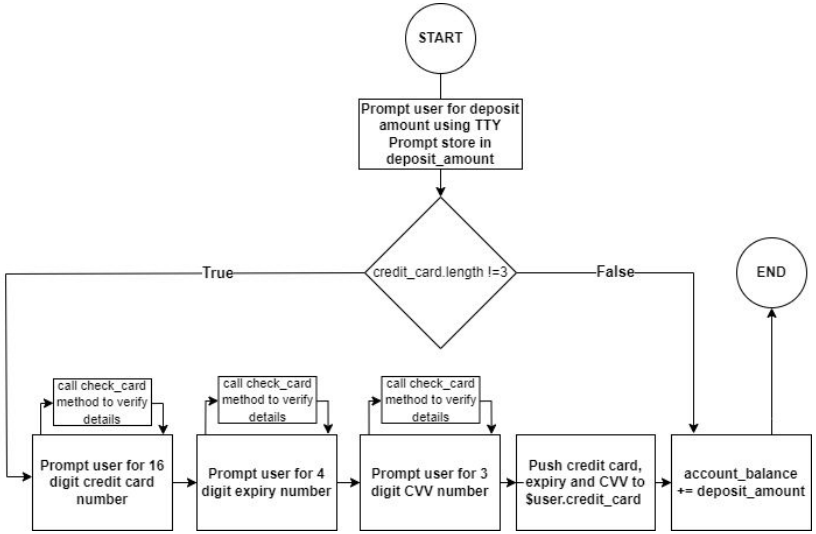


Feature 1 (Deposit Funds)

1. Gets deposit amount from the user
2. If card isn't already saved in user object it requests card number, expiry, & cvv
3. Adds card details to user's account
4. Updates user's account balance with deposit amount.

Handling Incorrect User Input

1. Calls check_card method that checks that card number inputs are in correct format.
2. If they aren't method requests relevant number again.



Feature 2 (Place Bet)

1. Allows users to select which fight they would like to place a bet on
2. Then which fighter they would like to bet on (along with associated odds)
3. Then how much they would like to wager
4. Then determines winner and updates account balance accordingly.

Refer to slides 16 - 17 for logical flow of place_bet method

Handling Incorrect User Input

Conditionals checks below, alerts user and exits to home menu if true.

- User account balance lower than min. bet ($1)
- Wager amount greater than max. bet ($10K) or less than min. bet.
- All fights have elapsed
- Wager amount greater than account balance

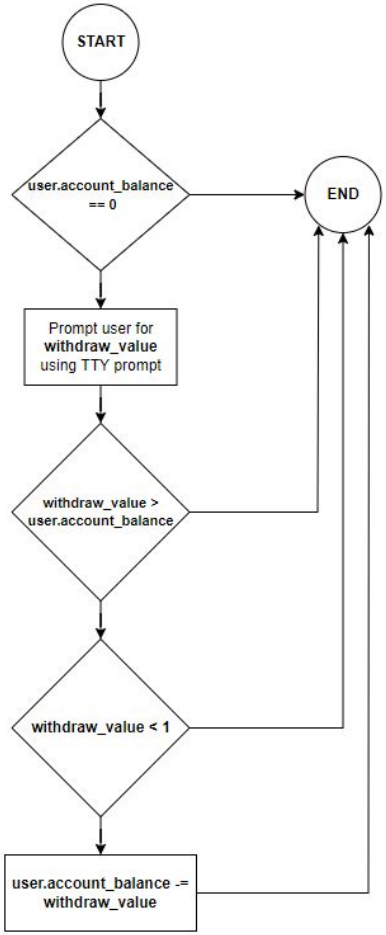
Feature 3 (Withdraw Funds)

1. Prompts user for withdraw amount
2. Subtracts withdraw value from account balance
3. Alerts user that transaction processed and print account balance.

Handling Incorrect User Input

Conditionals below are checked, alerts user and exits to home menu if true.

- User account balance == 0
- User withdraw value is greater than account balance
- User withdraw value is less than $1



Basic Logical Flow

Create User Account

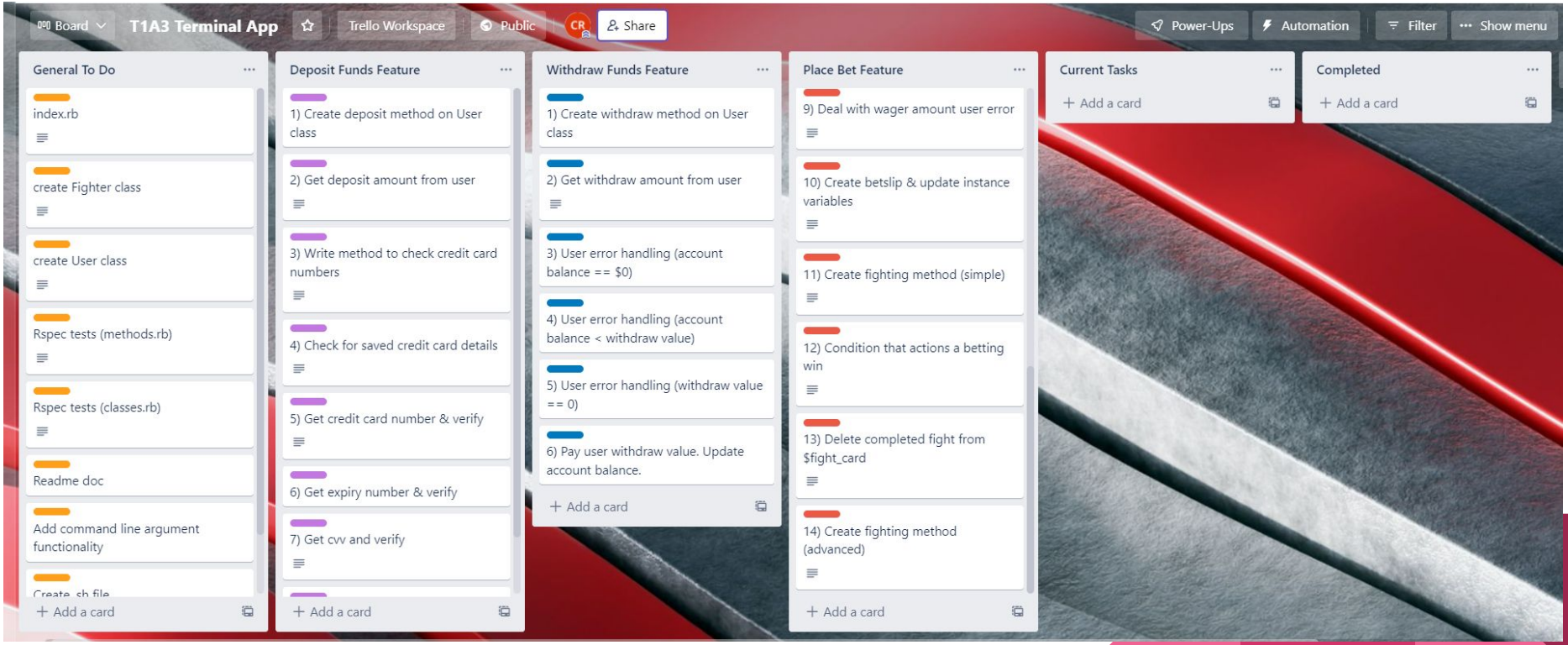
Deposit Funds

Place Bets

Withdraw Funds

Exit

Project Management - Trello Board



Challenges

Limited Time

Time limits resulted in the implementation of a scaled back version of **fighting** method. Which will be updated with more complexity after submission.

Bugs

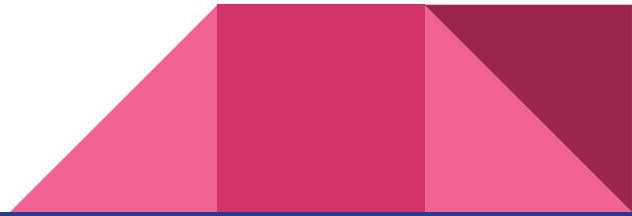
No major bugs, but several have taken a while to track down (especially within the place_bet method, which contains more complexity than the other methods)

Code

Project source code structure is split up into 3 main files:

1. **Classes.rb** - this houses the Fighter and User classes.
2. **Methods.rb** - this contains methods used for general functionality in the app
3. **Index.rb** - this creates all classes, and contains the main menu in a while loop, which is where the majority of the program runs from.

Rspec files are contained in the spec directory, and contain spec files that test the outputs of methods in Classes.rb and Methods.rb



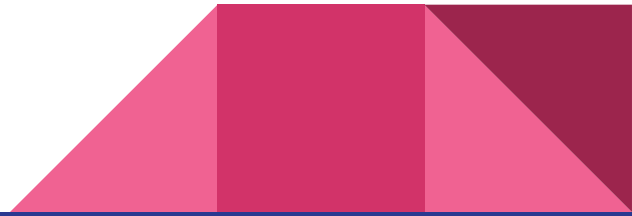
Ruby Gems

TTY Prompt
Used .select and .ask methods to collect user input.

TTY Font
Used to welcome and farewell users with large engaging font.

Colorize
Used in fighting method to color text green if user won bet, and red if they lost bet.

Faker
Used to generate fake first name, last name, and fighting scores in the Fighter class.



Code - Methods.rb

- **verify_name** method is passed user's first and last name and ensures it only contains valid characters and that it's not empty. Otherwise requests input again. Returns name if correct.

- **check _phone** method checks user's mobile number to ensure it only contains digits, it's not empty, that it starts with a valid mobile prefix, and that it's the correct amount of digits. Otherwise requests input again. Returns number if correct.

```ruby
# checks if name input provided is an empty string, or if it's got characters that aren't letters, hyphens, apostrophes, spaces.
def verify_name(name)
    begin
    raise InvalidCharacters if !name.count("^a-zA-Z' \-").zero? || name.empty?
    rescue InvalidCharacters
        puts "Please enter a valid name, using only letters, hyphens and apostrophes"
        name = STDIN.gets.strip
        retry
    end
    return name
end

# checks whether the phone number is a valid mobile number
def check_phone(phone_number)
    begin
        raise PhoneInvalid if !phone_number.count("^0-9").zero? || phone_number.empty? || phone_number.size != 10 || !phone_number.start_with?("04", "05")
    rescue PhoneInvalid
        puts "Please enter a valid 10 digit mobile phone number beginning with 04 or 05"
        phone_number = STDIN.gets.gsub(/\s+/, "")
        retry
    end
    return phone_number
end
```

Bash Script

- I've also included a shell script that users can use to run the application. This installs the ruby gem "bundler", then uses bundler to install all dependencies, then clears the terminal and runs the application.
- 3 command line arguments are passed in (first & last name and mobile number) which are used in the program to represent user account details.
- By default these arguments are my own details, but can be changed inside the program. The user can also pass their own details in via the terminal, or by editing the .sh file.

```
1   #!/bin/bash
2
3   #install bundler gem
4   gem install bundler
5
6   # #install gems required for the app
7   bundle install
8
9   clear
10
11  #run the application
12  ruby src/index.rb "Connor" "Roberts" "0407262636"
```

Code - Methods.rb

Both of these methods are required for, and called from, the upcoming place_bet method.

- **generate_odds** is passed a range (which is the sum of both fighters fighter scores), and the fighter score of one fighter. It returns the odds of that fighter winning.
- **fighting** is passed both fighters, and the fighter the user selected. It multiplies their fight scores by a random float in a narrow range and returns a winner based on that chance score. This simulates skill interacting with random chance.

```ruby
def generate_odds(range, fighter_score)
    if (range / fighter_score).round(2) < 1.10
        return 1.10
    else
        return (range / fighter_score).round(2)
    end
end
```

```ruby
def fighting(fighter_1, fighter_2, fighter_selected)
    fighter_1_chance_score = fighter_1.fighter_score * rand(0.6..1.0)
    fighter_2_chance_score = fighter_2.fighter_score * rand(0.6..1.0)

    fighter_1_chance_score > fighter_2_chance_score ? winner = fighter_1 : winner = fighter_2
    fighter_1_chance_score > fighter_2_chance_score ? loser = fighter_2 : loser = fighter_1
    sleep(2)
    if winner == fighter_selected
        puts "#{fighter_selected.full_name} won the fight! You won your bet!".colorize(:green)
        return winner
    else
        puts "#{fighter_selected.full_name} lost the fight. You lost your bet.".colorize(:red)
        return winner
    end
end
```

Code - Methods.rb

Place Bet Method Pt. 1

1) This checks whether user's account balance is less than min. bet, or if all fights have elapsed. If so, exits.

2) Then it prompts user to select an upcoming fight

3) Then matches their selection to the corresponding value in **$fight_array** (which tracks all 5 fights)

4) **Then** stores that fight's index, each fighter's object, the range, & each fighter's odds in a **fight hash** (calls generate_odds to populate odds).

5) **Then** prompts user to select a fighter from that specific fight to bet on (also displays their odds).

```ruby
def place_bet
    #identifies if user account balance is insufficient, or if there are no fights and exits.
    if $user.account_balance < 1.0
        system("clear")
        puts "You do not have sufficient credit on your account. Please make a deposit to place a bet."
    elsif $fight_card.size == 0
        system("clear")
        puts "There are no upcoming fights to place bets on."
    else
        #creates fight options to pass to tty prompt based on contents of $fight_card
        fight_options = []
        $fight_card.each do |i|
            fight_options.push("#{i[0].full_name} VS #{i[1].full_name}")
        end
        prompt = TTY::Prompt.new
        fight_select = prompt.select("Which upcoming fight would you like to bet on?", [fight_options, "Back"])
        system("clear")
        if fight_select == "Back"
            return
        else
            #creates a fight hash which will store the selected fighters, their odds, and their index in $fight_card.
            #then populates it with relevant data based on user's fight_select pick
            fight =  {}
            counter = 0
            while counter < $fight_card.size
                if fight_select.include? $fight_card[counter][0].full_name
                    fight[:fight_card_index] = counter
                    fight[:fighter_1] = $fight_card[counter][0]
                    fight[:fighter_2] = $fight_card[counter][1]
                    range = fight[:fighter_1].fighter_score + fight[:fighter_2].fighter_score
                    fight[:fighter_1_odds] = generate_odds(range, fight[:fighter_1].fighter_score)
                    fight[:fighter_2_odds] = generate_odds(range, fight[:fighter_2].fighter_score)
                end
                counter += 1
            end

            #creates a hash that we'll pass to tty prompt object to choose between the two fighters
            fighter_options = [
                {name: "#{fight[:fighter_1].full_name} at #{fight[:fighter_1_odds]} to 1", value: 1},
                {name: "#{fight[:fighter_2].full_name} at #{fight[:fighter_2_odds]} to 1", value: 2}
            ]
            fighter_choice = prompt.select("Which upcoming fight would you like to bet on?", [fighter_options, "Back"])
```

Code - Methods.rb

Place Bet Method Pt. 2.

1) Pushes the user's fighter choice, and their associated odds in the **fight hash**

2) **Then** it prompts user to input a wager amount.

3) If wager value is more than account balance, or not between $1 - $10K, then it alerts user and exits method.

4) Otherwise creates **betslip hash** with bet id, fighter_selected, wager, odds, and "won" keys, and pushes it to user's **bet_history** instance variable.

5) Then subtracts wager from account balance.

6) If fighting method returns the fighter the user selected, then it adds (wager * odds) to the user's account balance and updates "won" key in betslip to true.

7) Deletes the completed fight array from $fight_array, and tells user their account balance. Exits.

```ruby
if fighter_choice == "Back"
    system("clear")
    return
else
    #identifies the user's choice and stores their choice in fight hash.
    if fighter_choice == 1
        fight[:fighter_selection] = fight[:fighter_1]
        fight[:fighter_selection_odds] = fight[:fighter_1_odds]
    elsif fighter_choice == 2
        fight[:fighter_selection] = fight[:fighter_2]
        fight[:fighter_selection_odds] = fight[:fighter_2_odds]
    end

    #requests wager amount, if it exceeds account balance or is outside of the valid range it will exit back to start menu
    #otherwise if will continue execution in a nested if statement.
    wager_amount = prompt.ask("Enter dollar value of bet between $1 and $10,000", convert: :float) do |q|
        q.convert(:float, "%{value} is not a valid bet amount. Please enter a dollar value of bet using only numbers")
        q.required true
    end

    if wager_amount > $user.account_balance
        system("clear")
        puts "You do not have sufficient funds in your account to place this bet"
    elsif wager_amount > 10000 || wager_amount < 1
        system("clear")
        puts "You can only place bets between $1 and $10,000"
    else
        betslip = {
            :id => $user.bet_history.size,
            :fighter_selected => fight[:fighter_selection],
            :wager => wager_amount,
            :odds => fight[:fighter_selection_odds],
            :won => false
        }
        $user.bet_history.push(betslip)
        $user.account_balance -= betslip[:wager]

        if fighting(fight[:fighter_1], fight[:fighter_2], betslip[:fighter_selected]) == betslip[:fighter_selected]
            $user.account_balance += (betslip[:wager] * betslip[:odds]).round(2)     #this updates the account balance if they won.
            $user.bet_history[betslip[:id]][:won] = true    #this updates the won key to true if they won
        end
        $fight_card.delete_at(fight[:fight_card_index])   #deletes the relevant $fight_card array.
        puts "Your account balance: #{$user.account_balance}."
    end
end
```

Code - Classes.rb

Custom error classes

- Created InvalidCharacters, PhoneInvalid, and CardInvalid error classes for use in verification methods across methods.rb and classes.rb

```ruby
class InvalidCharacters  < StandardError
end

class PhoneInvalid < StandardError
end

class CardInvalid < StandardError
end
```

Fighter Class

- Sets instance variables (name properties, fighting scores, and damage)
- Defines set_fighter_score method - which sets the fighter_score

```ruby
class Fighter
    attr_reader :first_name, :last_name, :grappling_score, :striking_score, :power_score, :full_name
    attr_accessor :damage, :fighter_score
    def initialize
        @first_name = Faker::Name.male_first_name
        @last_name = Faker::Name.last_name
        @full_name = "#{@first_name} #{@last_name}"
        @grappling_score = Faker::Number.within(range: 3.0..10.0)
        @striking_score = Faker::Number.within(range: 3.0..10.0)
        @power_score = Faker::Number.within(range: 3.0..10.0)
        @fighter_score = 0.0
        @damage = 0
    end

    def set_fighter_score
        @fighter_score = @grappling_score +  @striking_score + @power_score
        return @fighter_score
    end
end
```

Code - Classes.rb

User Class pt. 1

Initialize method

- Passed first & last name, and phone number. Sets instance variables (name, phone number, account balance, credit card details, and bet history properties).

```ruby
class User
    attr_reader :first_name, :last_name, :phone_number, :account_balance, :credit_card
    attr_accessor :bet_history, :account_balance
    def initialize(f_name, l_name, phone_no)
        @first_name =  f_name
        @last_name =  l_name
        @phone_number =  phone_no
        @account_balance =  0
        @credit_card = []
        @bet_history = []
    end
```

cash_out (User method)

1. If account balance is 0, alert user & exit method
2. Prompts user to provide withdraw value, converts this number to a float
3. If withdraw value is greater than account balance, or withdraw value is less than $1, then we alert user and exit method.
4. Otherwise we subtract withdraw_value from user's account balance and advise them we've processed transaction.

```ruby
def cash_out
    if @account_balance == 0
        system("clear")
        puts "Your account balance is $0. You cannot withdraw funds."
    else
        prompt = TTY::Prompt.new
        puts "Your account balance is #{@account_balance.round(2)}"
        withdraw_value = prompt.ask("How much would you like to withdraw? Enter a number.", convert: :float, required: true) do |q|
        q.convert(:float, "%{value} is not a valid withdrawal amount. Please enter a number.")
        end

        withdraw_value = withdraw_value.round(2)

        if withdraw_value > @account_balance
            system("clear")
            puts "Your requested cash out value of #{withdraw_value} is greater than your account value."
        elsif withdraw_value < 1
            system("clear")
            puts "You cannot withdraw less than $1 from your account balance."
        else
            @account_balance -= withdraw_value
            system("clear")
            puts "We have transferred $#{withdraw_value} to the pay id associated with the phone number #{@phone_number}."
            puts "Your remaining account balance is $#{@account_balance.round(2)}"
        end
    end
end
```

Code - Classes.rb

Check_card (User method)

- Method is passed card number and expected digits, then returns the number if it contains only numbers, isn't empty, and it's size is equal to the digits value. Otherwise requests card number again.

Deposit (User method)

1. Prompts user to select deposit amount.
2. Checks if card details are already stored in user object
3. If not it will request card number, expiry, & cvv. For each it will call check_card (passing in provided number & expected digits) to verify input, then it stores the card number as integer in user's credit_card property.
4. Adds deposit amount to user's account balance, alerts user, and exits method.

```ruby
def check_card(number, digits)
    begin
        raise CardInvalid if !number.count("^0-9").zero? || number.empty? || number.size != digits
    rescue CardInvalid
        puts "Please enter a valid #{digits} digit number"
        number = STDIN.gets.gsub(/\s+/, "")
        retry
    end
    return number
end

def deposit
        prompt = TTY::Prompt.new
        deposit_amount = prompt.select("Please enter deposit amount", ["5", "10", "20", "35", "50", "75", "100"]).to_f

        if @credit_card.length != 3
        puts "Please enter your 16 digit credit card number"
        credit_card_no = STDIN.gets.gsub(/\s+/, "")
        check_card(credit_card_no, 16)
        credit_card_no = credit_card_no.to_i
        @credit_card[0] = credit_card_no

        puts "Please enter your 4 digit expiry date"
        expiry_date = STDIN.gets.gsub(/\s+/, "")
        check_card(expiry_date, 4)
        expiry_date = expiry_date.to_i
        @credit_card[1] = expiry_date

        puts "Please enter your 3 digit CVV"
        cvv = STDIN.gets.gsub(/\s+/, "")
        check_card(cvv, 3)
        cvv = cvv.to_i
        @credit_card[2] = cvv
    end
    @account_balance += deposit_amount
    system("clear")
    puts "Transaction successful. #{deposit_amount} deposited. Your account balance is #{@account_balance}"
end
```

Code - Index.rb pt. 1

1. Welcomes user to the app with tty-font graphic.
2. Checks if 3 command line arguments are provided (first & last name, mobile number)
3. If they are then it asks user if account details are correct.
4. If they respond “No”, or if no command line arguments aren’t provided, then it prompts user for these details and uses verify_name & check_phone methods to verify data.
5. Creates user object from User class using data collected from prompts, or from command line arguments if provided and if user selected “Yes”.
6. Clears terminal, welcomes user using their name, and pauses 2 seconds.

```ruby
#writes app name in stylised writing
font = TTY::Font.new(:doom)
puts font.write("WELCOME TO")
puts font.write("SMESH BETS")

#create tty prompt object
prompt = TTY::Prompt.new

#ARGV tests
command_line_login = "No"

if ARGV.size == 3
    puts "You have provided the following account details:"
    puts "Name: #{ARGV[0]} #{ARGV[1]}"
    puts "Phone: #{ARGV[2]}"
    command_line_login = prompt.select("Is this correct?", ["Yes", "No"])

    if command_line_login == "Yes"
        first_name = ARGV[0] if ARGV[0]
        last_name = ARGV[1] if ARGV[1]
        phone_number = ARGV[2] if ARGV[2]
        first_name = verify_name(first_name)
        last_name = verify_name(last_name)
        phone_number = check_phone(phone_number)
        system("clear")
    end
end

if command_line_login == "No"
    system("clear")
    puts "Please enter your details to register an account"
    first_name = prompt.ask("What is your first name?", required: true)
    first_name = verify_name(first_name)
    last_name = prompt.ask("What is your last name?", required: true)
    last_name = verify_name(last_name)
    phone_number = prompt.ask("Please enter a 10 digit mobile number beginning with 04 or 05", required: true).gsub(/\s+/, "")
    phone_number = check_phone(phone_number)
end

$user = User.new(first_name, last_name, phone_number)

system("clear")
puts "Welcome #{$user.first_name}!"
sleep(2)
```

Code - Index.rb pt. 2

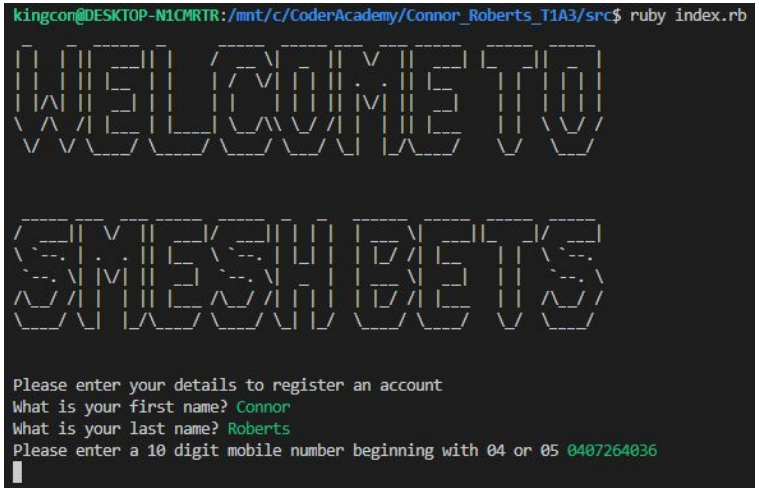
Index.rb file pt. 2

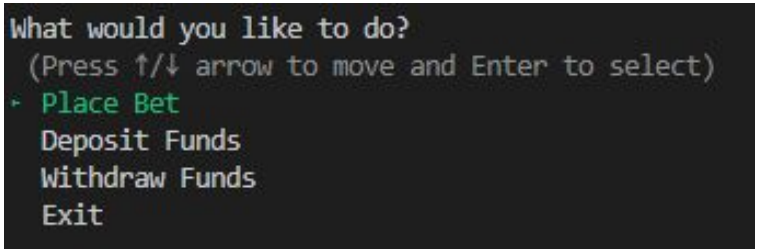
1. Creates a $fight_card array where the 5 upcoming fights are stored as arrays (2 new Fighter objects in each).
2. Set's the fighter score for all new fighters.
3. Initiates a while loop which contains the home menu which provides navigation to all functionality, and is where the program executes from until 'Exit' is selected.

```ruby
#Creates all fighters
$fight_card = [[Fighter.new, Fighter.new], [Fighter.new, Fighter.new], [Fighter.new, Fighter.new], [Fighter.new, Fighter.new], [Fighter.new, Fighter.new]]

#sets their fighter_scores using the Fighter class method.
$fight_card.each do |i|
    i[0].set_fighter_score
    i[1].set_fighter_score
end


#home menu
system("clear")
program_active = true
puts "What would you like to do?"
while program_active
home_menu_select = prompt.select("", ["Place Bet", "Deposit Funds", "Withdraw Funds", "Exit"])
    if home_menu_select == "Place Bet"
        place_bet
    elsif home_menu_select == "Deposit Funds"
        $user.deposit
    elsif home_menu_select == "Withdraw Funds"
        $user.cash_out
    elsif home_menu_select == "Exit"
        program_active = false
        system("clear")
        puts font.write("GOODBYE")
    end
end
```

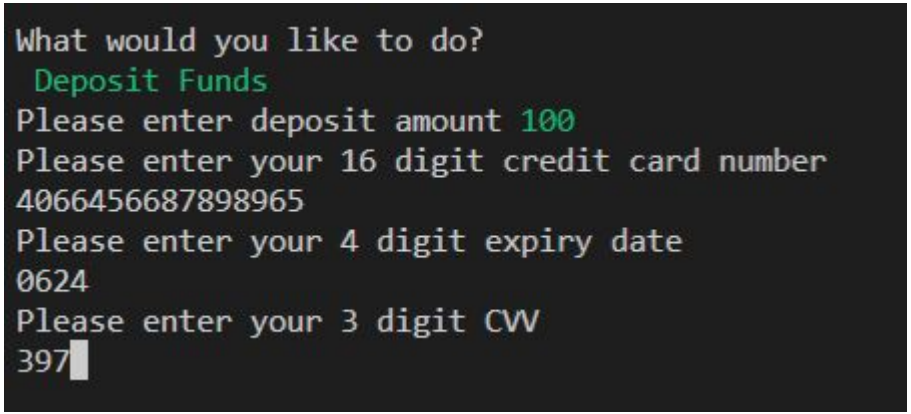
App Functionality

Welcome - Create User - Main Menu





Deposit Funds



App Functionality

Place Bet

```
Place Bet
Which upcoming fight would you like to bet on? (Press ↑/↓ arrow to move and Enter
‣ Larry Carter VS Lenny Powlowski
  Adolfo Cruickshank VS Stan Blanda
  Cliff Powlowski VS Bo Greenfelder
  Tobias Orn VS Lincoln Cummerata
  Levi Homenick VS Darrick Daniel
  Back
```

```
Which fighter would you like to bet on?
‣ Levi Homenick at 2.24 to 1
  Darrick Daniel at 1.81 to 1
  Back
```

```
Which upcoming fight would you like to bet on? Levi Homenick at 2.24 to 1
Enter dollar value of bet between $1 and $10,000 50
Levi Homenick lost the fight. You lost your bet.
Your account balance: 50.0.
```

Withdraw Funds

```
Withdraw Funds
How much would you like to withdraw? Enter a number. 150
We have transferred $150.0 to the pay id associated with the phone number 0407264036.
Your remaining account balance is $50.0
```