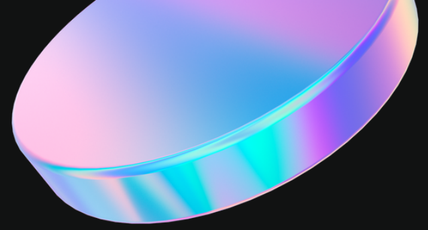
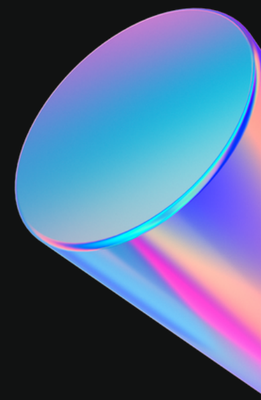
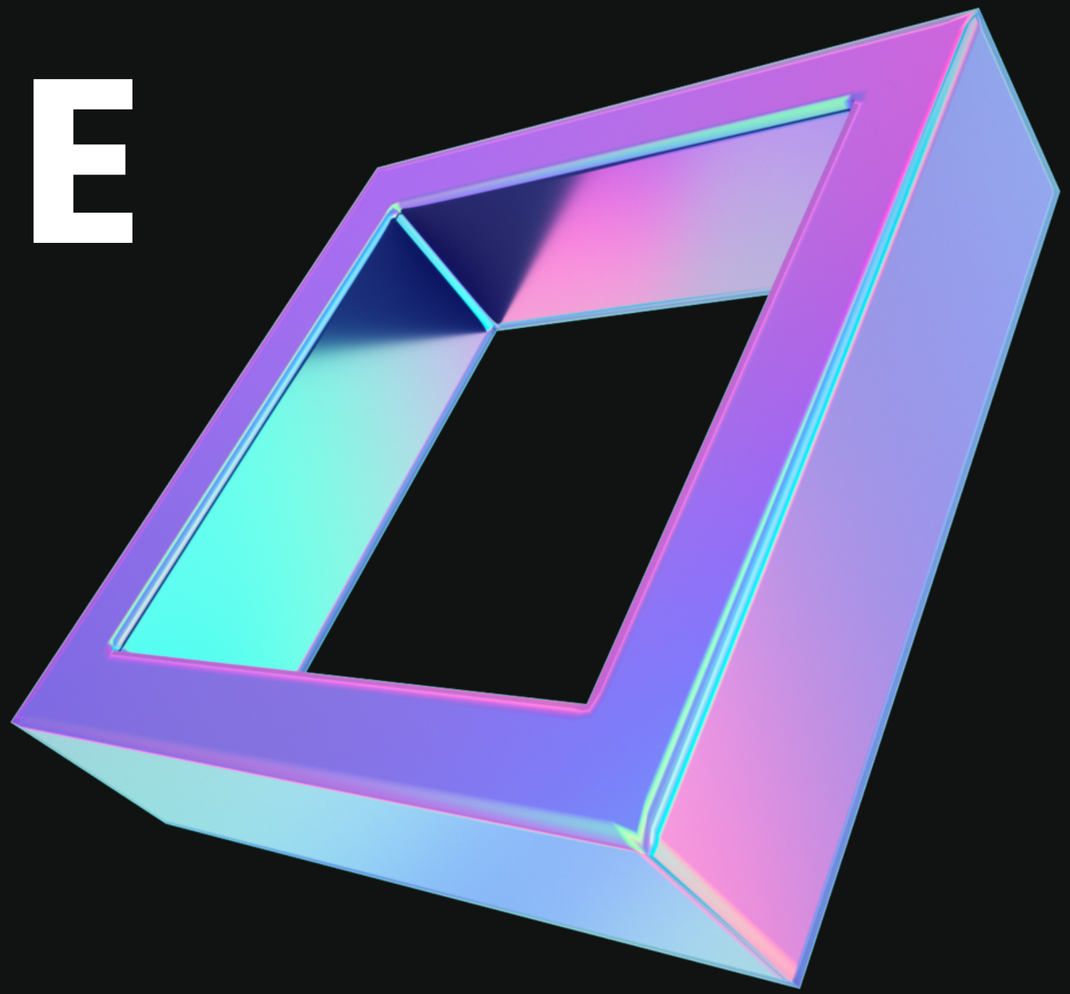
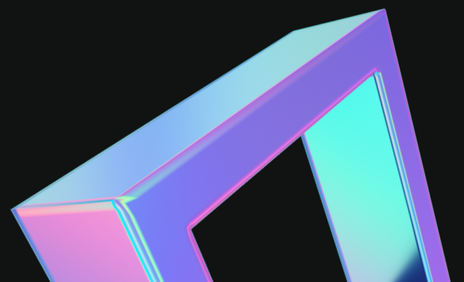
RIDE THE BUS

Terminal Application









Overview of Application

GAME

Game is a card game comprised of four levels, each require some logic and probability analysis

MAIN MENU

Main menu used at the primary navigation page for all aspects of the app

SCOREBOARD + PUNISHMENTS

A scoreboard keeps track of each players name and the number of turns they took to complete the game. Once all players have finished playing, the game will display the loser and a random punishment!

```ruby
case input.to_i
when 0
    display_menu()
when 1
    Game.game_start()
    display_menu()
    system("clear")
    puts "Returning to Main Menu..."
when 2
    Instructions.instructions()
    puts "Press enter to return to main menu...".colorize(:blue)
    gets
    system("clear")
    display_menu()
when 3
    a = Artii::Base.new
    puts a.asciify("Hall of Shame !").colorize(:cyan)
    puts "Here are the current the scores"
    Game.display_scores()
    Punishments.display_punishments(punishments)
    puts "Press enter to return to main menu...".colorize(:blue)
    gets
    system("clear")
    display_menu()
when 4
    puts Game.display_lowest()
    puts Punishments.random_punshiment(punishments)
    puts "\nPress enter to return to main menu...".colorize(:blue)
    gets
    system("clear")
    display_menu()
when 5
    puts "Goodbye, thanks for playing!"
    exit(0)
end
```

Main Menu

Code Snippet

- Requests User Input
- Converts string input into integer
- Validator checks if input is valid
  - If not, displays error and prompts user to try again
- Depending on the number input, different module functions will be called
- All options will eventually lead back to the main menu
  - App can also be exited through the main menu

```ruby
module Validator

    def self.validate_input(input)
      num_input = input.to_i
      if num_input > 0 && num_input < 6
        return true
      else
        return false
      end
    end

end
```

```ruby
require_relative '../controller/validator'

describe 'validates' do
    it 'checks if input is valid' do
        expect(Validator.validate_input("1")).to eq(true)
        expect(Validator.validate_input("2")).to eq(true)
        expect(Validator.validate_input("3")).to eq(true)
        expect(Validator.validate_input("4")).to eq(true)
        expect(Validator.validate_input("5")).to eq(true)
        expect(Validator.validate_input("6")).to eq(false)
        expect(Validator.validate_input("7")).to eq(false)
        expect(Validator.validate_input("8")).to eq(false)
        expect(Validator.validate_input("9")).to eq(false)
        expect(Validator.validate_input("abc")).to eq(false)
        expect(Validator.validate_input("a2c4")).to eq(false)
        expect(Validator.validate_input("")).to eq(false)
    end
end
```

Main Menu

Error Handling + Testing

VALIDATOR

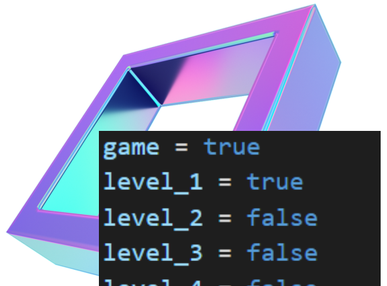
Validator module was used to test whether the user input for the main menu was valid

RSPEC

Rspec testing was used to make sure main menu items would return the correct truesy or falsey's

Main Menu

DEMO



```
game = true
level_1 = true
level_2 = false
level_3 = false
level_4 = false
play_again = false
```

Game

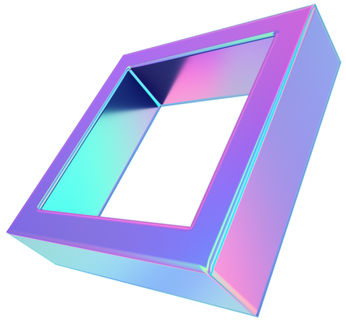
```
while game == true
    while level_1 == true
```

```
if red_or_black == symbol_colour[suit]
    puts card
    puts "\nCorrect!".colorize(:green)
    level_1 = false
    level_2 = true
    next
elsif red_or_black == 'Quit'
    puts "You have quit the game, your score has been set to automatically 100"
    level_1 = false
    play_again = true
    @counter = 100
    @scoreboard[name] = @counter
    @counter = 0
else
    puts card
    puts "\nIncorrect...".colorize(:red)
    @counter += 1
    puts "You have so far taken #{@counter} turns..."
end
```

```
while level_2 == true
```

Code Snippet

- Split between 4 levels, which are contained within a while loop
  - Level 1 - Red or Black
  - Level 2 - Higher or Lower
  - Level 3 - Inside or Outside
  - Level 4 - Suit
- Attempts initially set to 1
  - Can quit at any level, but attempts variable will be set to 100 if the player quits
- If incorrect, go back to level 1
  - Attempts += 1
- If successfully pass all 4 levels
  - Add name and corresponding attempt number to a scoreboard hash



Game

Error Handling + Testing

RSPEC

Make sure that the random cards being selected are unique, in both suits and numbers

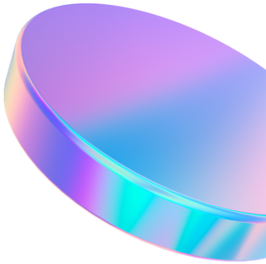
GEM - TTYPROMPT

Using tty-prompt to reduce the chance of an error occuring from typing an incorrect option.

```ruby
require_relative '../model/game'


describe 'card number' do
    it 'picks a key in a hash' do
        expect(Game.random_key({'key' => 'value'})).to eq('key')
        expect(Game.random_key({'key1' => 'value 1', 'key2' => 'value2'})).to eq('key1').or(eq('key2'))
    end
end

describe 'select suit' do
    it 'picks a suit from the array' do
        expect(Game.random_suit(["A"])).to eq("A")
        expect(Game.random_suit(["A", "B"])).to eq("A").or(eq("B"))
        expect(Game.random_suit(["♠", "♦", "♥", "♣"])).to eq("♠").or(eq("♦")).or(eq("♥")).or(eq("♣"))
    end
end
```



Ride the Bus Game

DEMO

```ruby
module Game
    @counter = 1
    @scoreboard = {}
```

```ruby
elsif what_suit == "diamonds" && random_suit4 == "♦"
    puts card4
    puts "\nCorrect!".colorize(:green)
    puts "Congratulations! You've won the game!".colorize(:green)
    level_4 = false
    play_again = true
    @counter +=1
    @scoreboard[name] = @counter
    @counter = 0
    next
elsif what_suit == 'Quit'
    puts "You have quit the game, your score has been set to automatically 100"
    level_4 = false
    play_again = true
    @counter = 100
    @scoreboard[name] = @counter
    @counter = 0
else
    puts card4
    puts "\nIncorrect...".colorize(:red)
    level_4 = false
    level_1 = true
    @counter += 1
    puts "You have so far taken #{@counter} turns..."
end
```

Scoreboard

Code Snippet

- Initially set as an instance variable within the Game module
  - Attempts counter
  - Scoreboard hash
- For each level that is incorrect, counter is set to += 1
- If user quits, their attempt counter is set to a default of 100
  - Added to the scoreboard hash
- If user finishes all 4 levels, they current counter will be added to the hash
- Scoreboard can be accessed through main menu

Scoreboard

Error Handling + Testing

TESTING

Handling the scoreboard in case it was empty, or in case there were multiple users with the same score

```
def display_lowest()
    if @scoreboard.empty?
        system("clear")
        a = Artii::Base.new
        puts a.asciify("Judgement Time !").colorize(:magenta)
        puts "The scoreboard is currently empty...".colorize(:red)
    else
        system("clear")
        a = Artii::Base.new
        puts a.asciify("Judgement Time !").colorize(:magenta)
        maximum = @scoreboard.values.max
        loser_count = @scoreboard.values.count(maximum)
        if loser_count > 1
            losers = @scoreboard.select {|key,value| value >= maximum}
            losers.each {|k,v| puts "#{k.capitalize} took #{v} turns to complete the game..."}
            puts "Unfortunately they'll have to do the punishment..."
        else
            puts "The person who took the most number of turns was #{scoreboard.key(maximum)}"
        end
    end
end
```

Scoreboard + Punishments

DEMO

Review of Development Process

CHALLENGES

- Making the logic statements for Level 3 to work
- Testing using Rspec
- Finding the right Ruby Gems to use
  - Figuring out how to use some Gems which had limited documentation

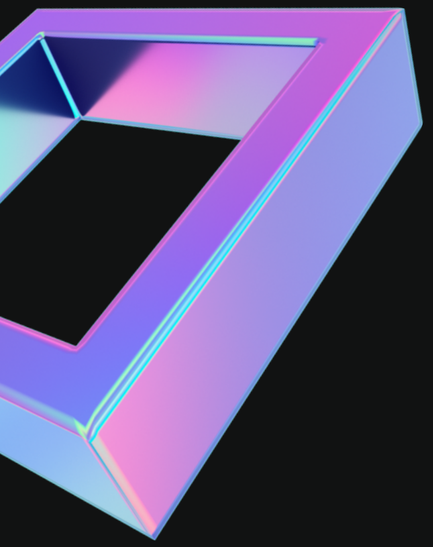
FAVOURITE PARTS

- Using Gems like artii, colorize and terminal-table to make the application look more aesthetic
- Making a game I've actually had fun playing before
- Actually playing the game during my testing to make sure the logic statements work

ETHICAL ISSUES

- None noted


Any
questions?



Thanks for listening!