Hangman by Ruby and Grey

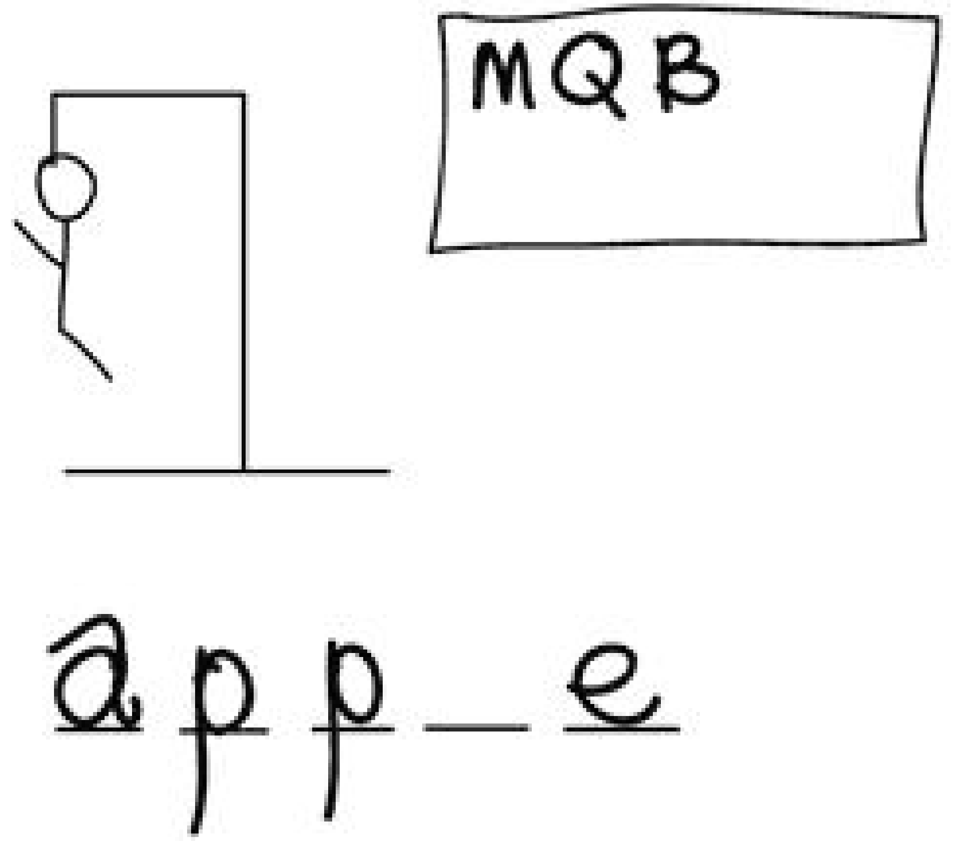
Billy Vanderlaar, Kelly McCleese, Sayeed Siddiqui

Overview

- What is Hangman?
- Controllers
- User implementation
- Gameplay
- Challenges
- Possible additions

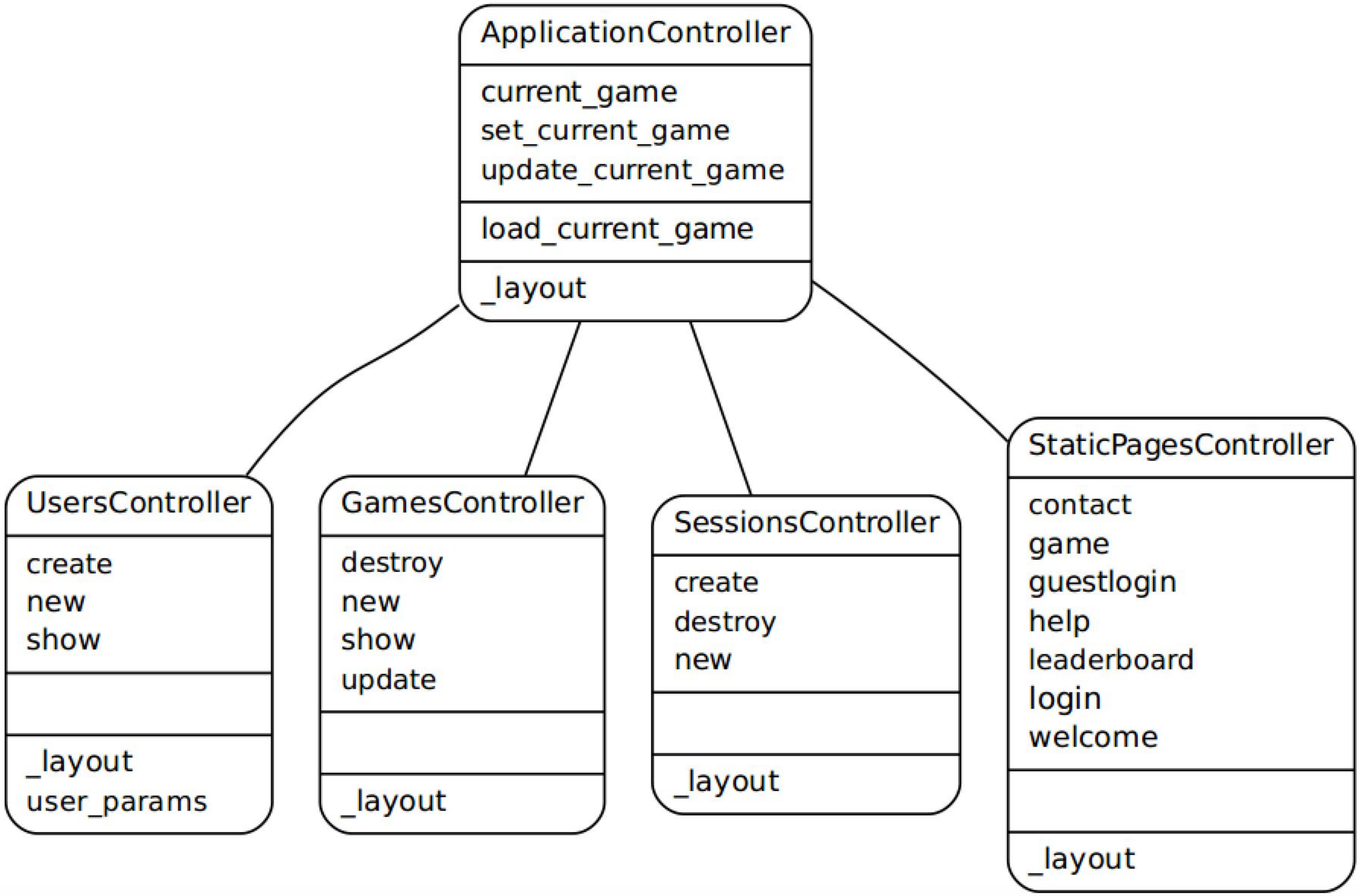
What is hangman

- A player picks a secret word
- They write out a number of dashes equal to the length of the word
- The other player guesses letters they think might be in the word
- If that letter is in the word, they write it on the correct dash
- If it is not in the word, a part of hangman is drawn until they run out of guesses



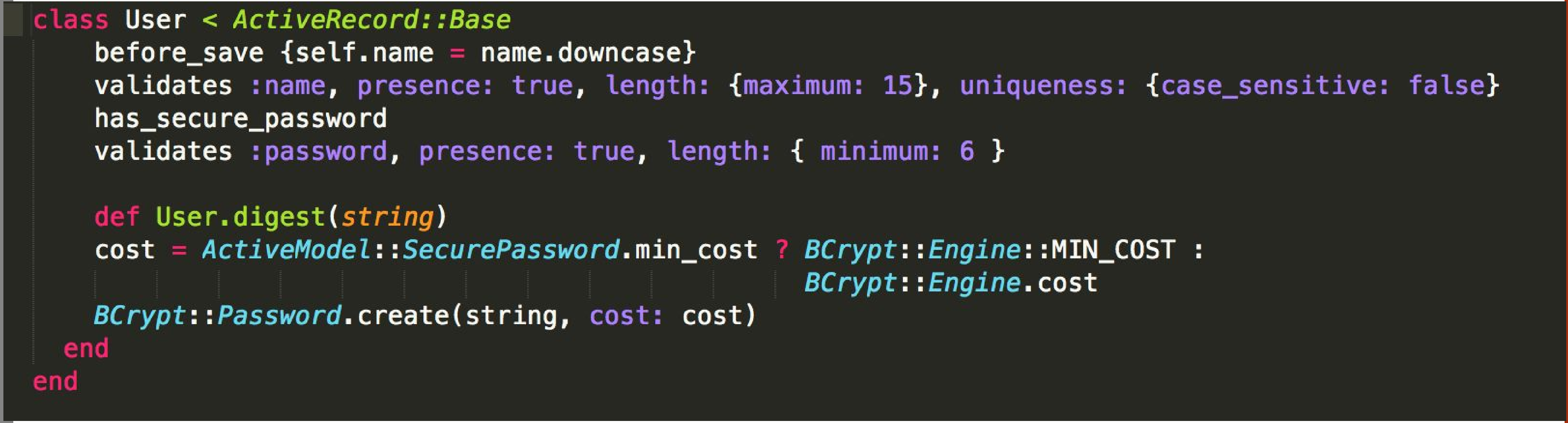
http://www.grandparents.com/grandkids/activities-games-and-crafts/hangman

Controllers



User Implementation

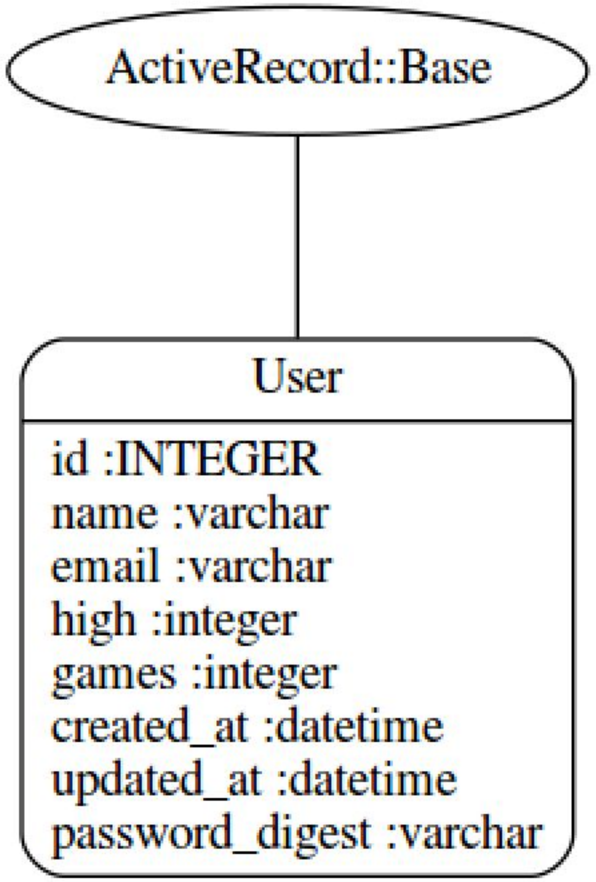
- Each username must be unique with maximum of 15 characters
- Each password must be at least 6 characters
- User Controller:



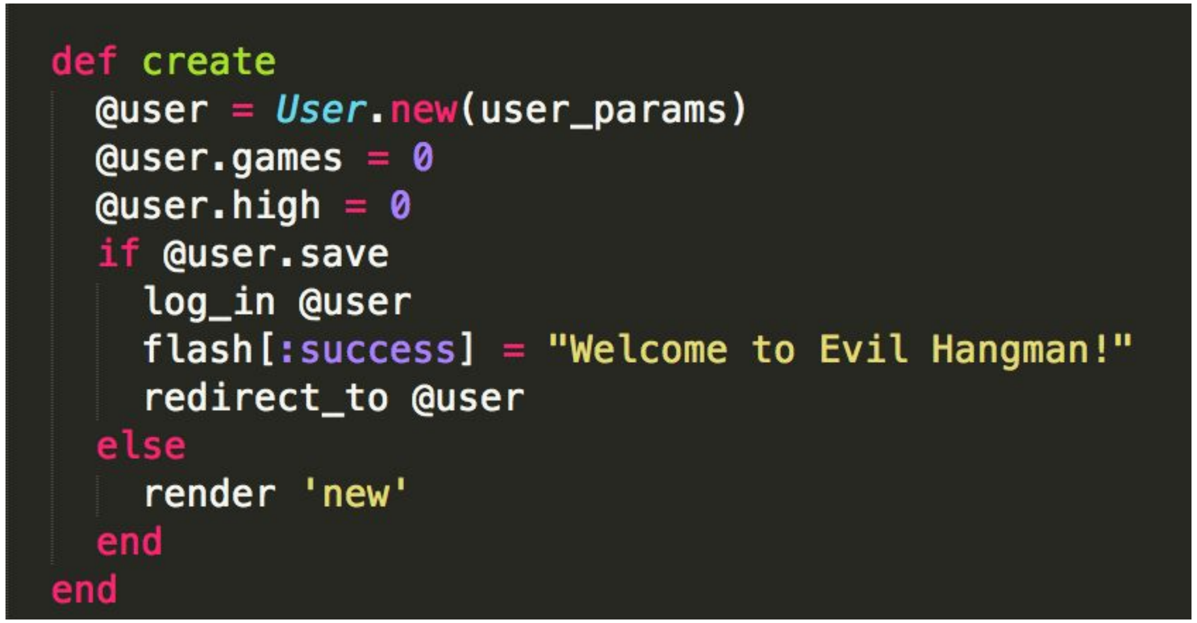
```ruby
class User < ActiveRecord::Base
    before_save {self.name = name.downcase}
    validates :name, presence: true, length: {maximum: 15}, uniqueness: {case_sensitive: false}
    has_secure_password
    validates :password, presence: true, length: { minimum: 6 }

    def User.digest(string)
    cost = ActiveModel::SecurePassword.min_cost ? BCrypt::Engine::MIN_COST :
                                                  BCrypt::Engine.cost
    BCrypt::Password.create(string, cost: cost)
  end
end
```

User Implementation

Controller picture:



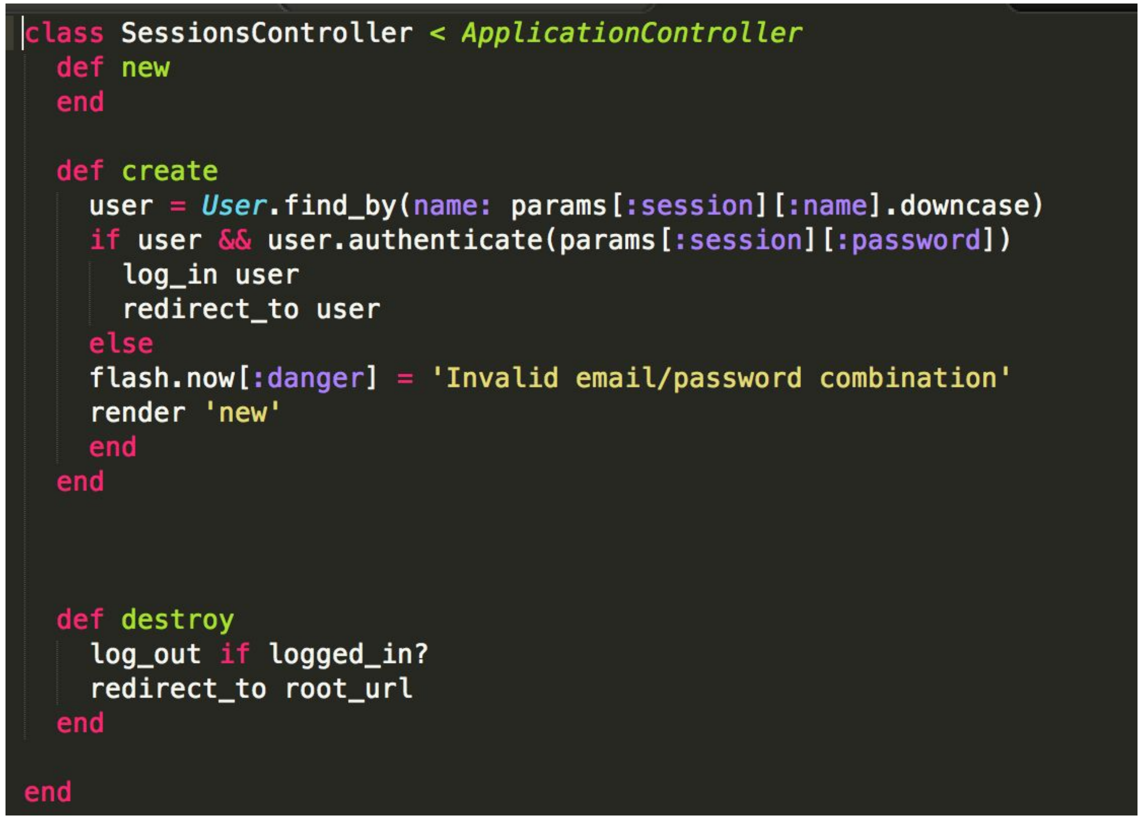
Create new user (or login):



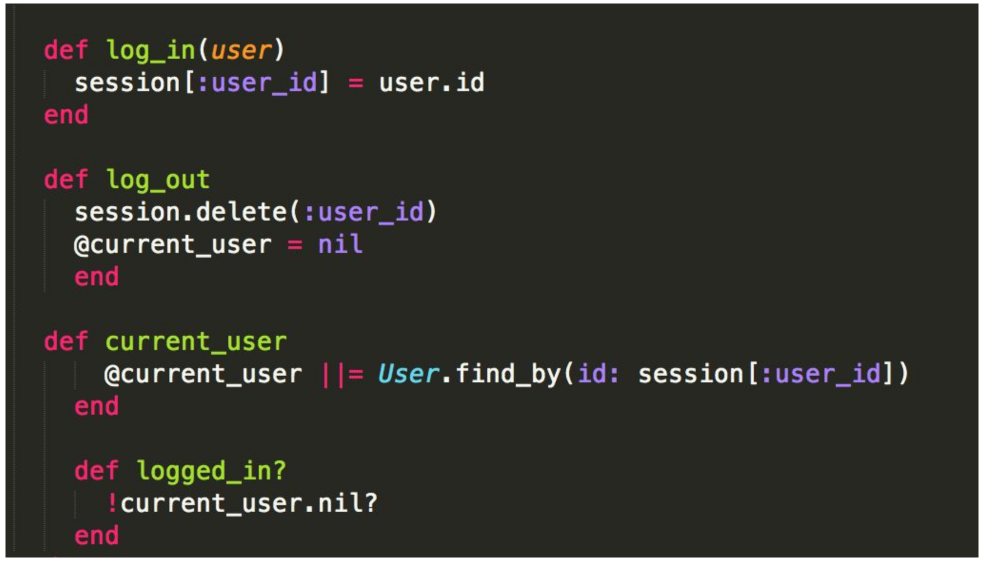
```ruby
def create
  @user = User.new(user_params)
  @user.games = 0
  @user.high = 0
  if @user.save
    log_in @user
    flash[:success] = "Welcome to Evil Hangman!"
    redirect_to @user
  else
    render 'new'
  end
end
```

Login/out

- Creates new user or validates existing user
- Displays error message if username/password incorrect
- Logs in user and navigates to user name



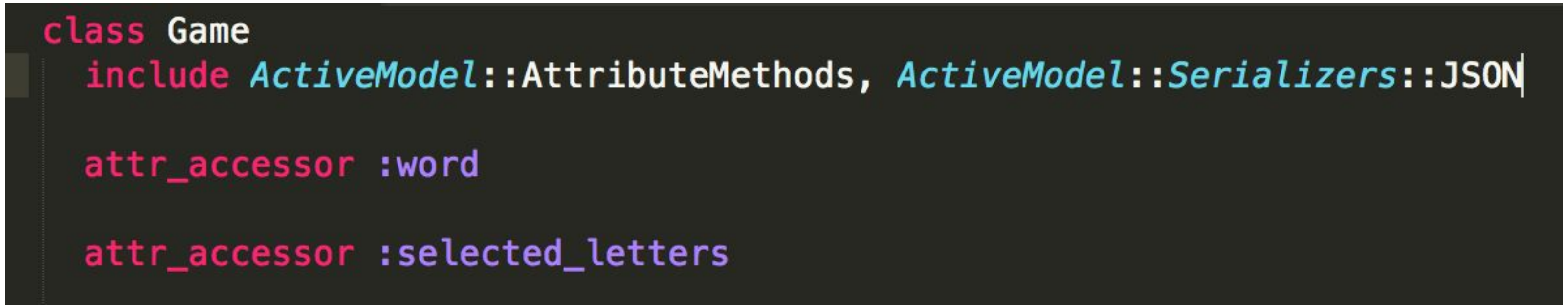
```ruby
class SessionsController < ApplicationController
  def new
  end

  def create
    user = User.find_by(name: params[:session][:name].downcase)
    if user && user.authenticate(params[:session][:password])
      log_in user
      redirect_to user
    else
    flash.now[:danger] = 'Invalid email/password combination'
    render 'new'
    end
  end



  def destroy
    log_out if logged_in?
    redirect_to root_url
  end

end
```



```ruby
def log_in(user)
  session[:user_id] = user.id
end

def log_out
  session.delete(:user_id)
  @current_user = nil
  end

def current_user
    @current_user ||= User.find_by(id: session[:user_id])
  end

  def logged_in?
    !current_user.nil?
  end
```

Gameplay

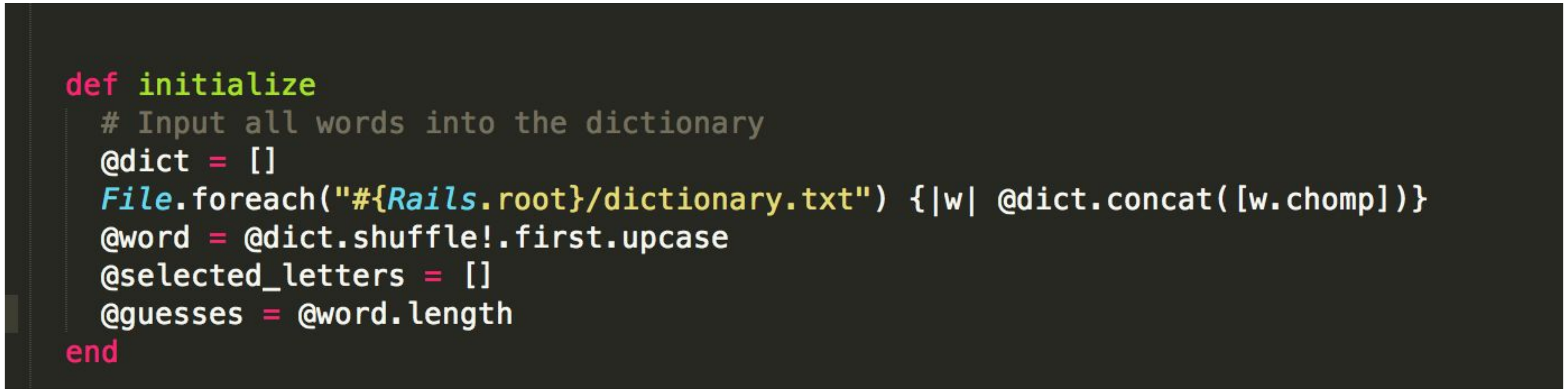
- Relatively simple concept
  - Computer picks a word
  - Player guesses letters in the word
  - Number of guess = word.length



```
class Game
  include ActiveModel::AttributeMethods, ActiveModel::Serializers::JSON

  attr_accessor :word

  attr_accessor :selected_letters
```

Gameplay:
How do you select a word?

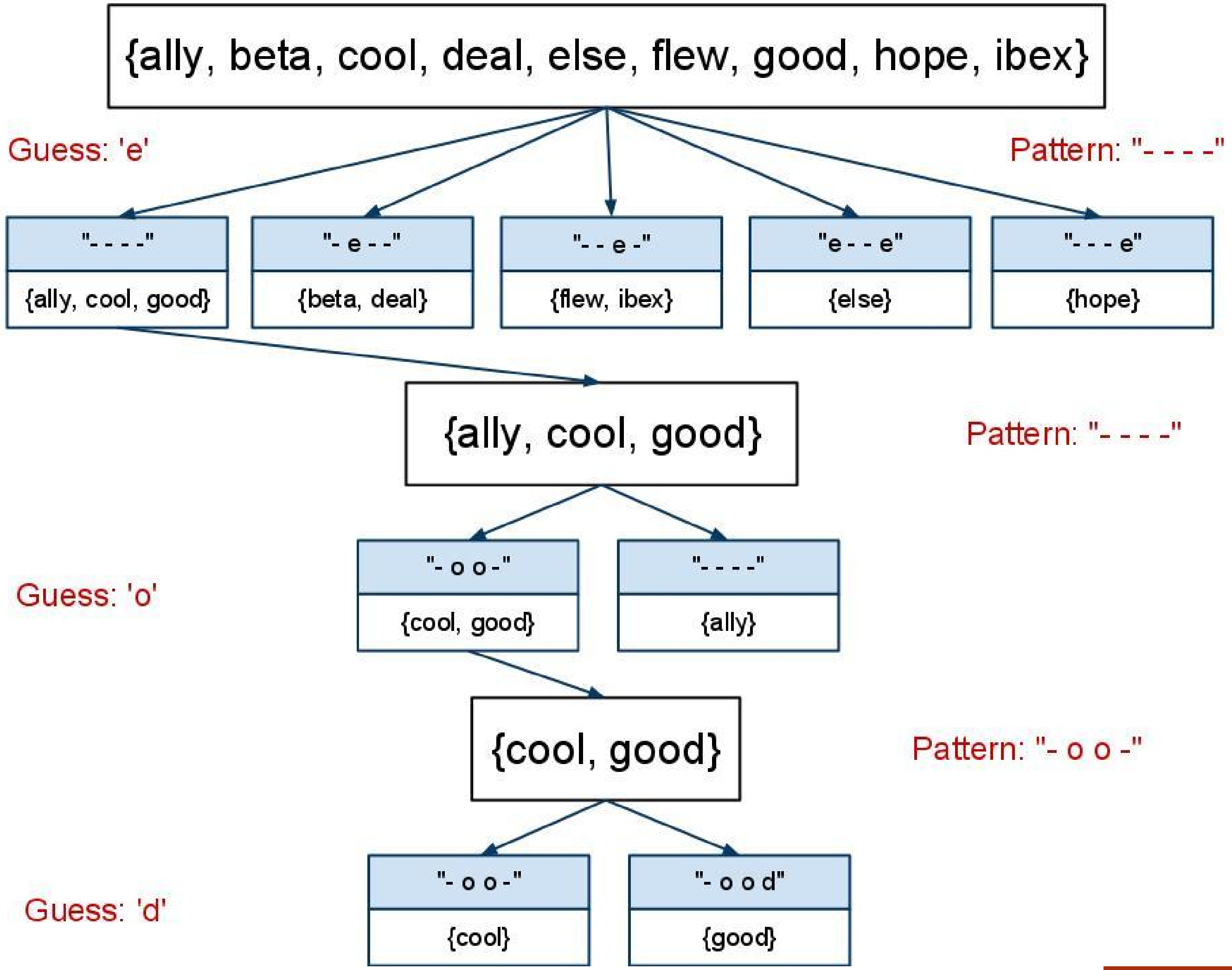
- File containing all possible words dictionary.txt located in root folder
- Create array containing all possible words, shuffle and select first



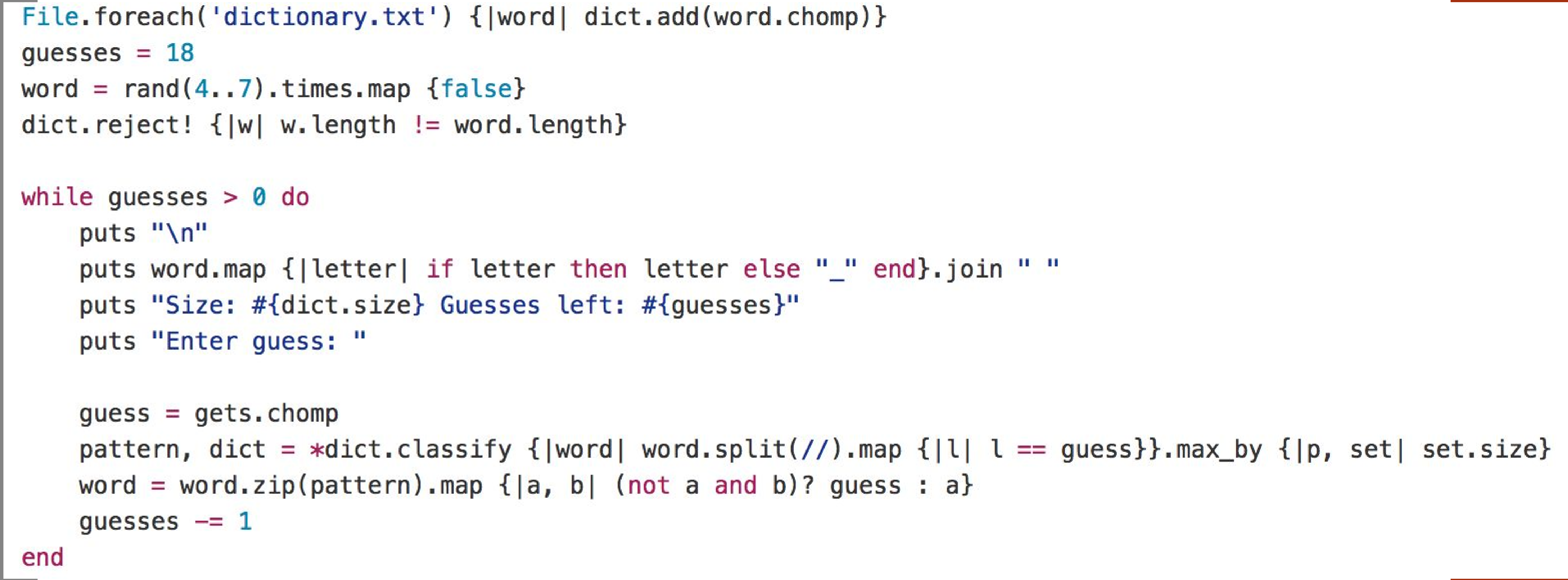
```ruby
def initialize
  # Input all words into the dictionary
  @dict = []
  File.foreach("#{Rails.root}/dictionary.txt") {|w| @dict.concat([w.chomp])}
  @word = @dict.shuffle!.first.upcase
  @selected_letters = []
  @guesses = @word.length
end
```

Why is your hangman evil?

- We actually were going to implement "evil hangman"
- Instead of a single word, the computer picks a list of words and dodges you as you guess letters
- The challenge of serialization on the previous slide was the reason we were unable to implement this type of hangman



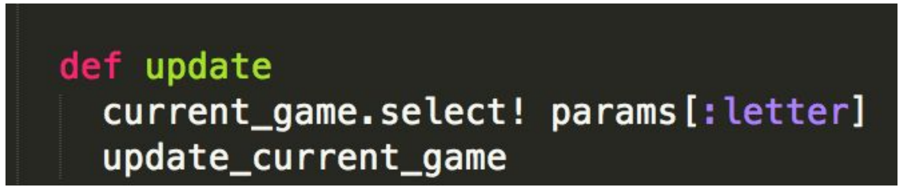
Evil Hangman Algorithm



```ruby
File.foreach('dictionary.txt') {|word| dict.add(word.chomp)}
guesses = 18
word = rand(4..7).times.map {false}
dict.reject! {|w| w.length != word.length}

while guesses > 0 do
    puts "\n"
    puts word.map {|letter| if letter then letter else "_" end}.join " "
    puts "Size: #{dict.size} Guesses left: #{guesses}"
    puts "Enter guess: "

    guess = gets.chomp
    pattern, dict = *dict.classify {|word| word.split(//).map {|l| l == guess}}.max_by {|p, set| set.size}
    word = word.zip(pattern).map {|a, b| (not a and b)? guess : a}
    guesses -= 1
end
```

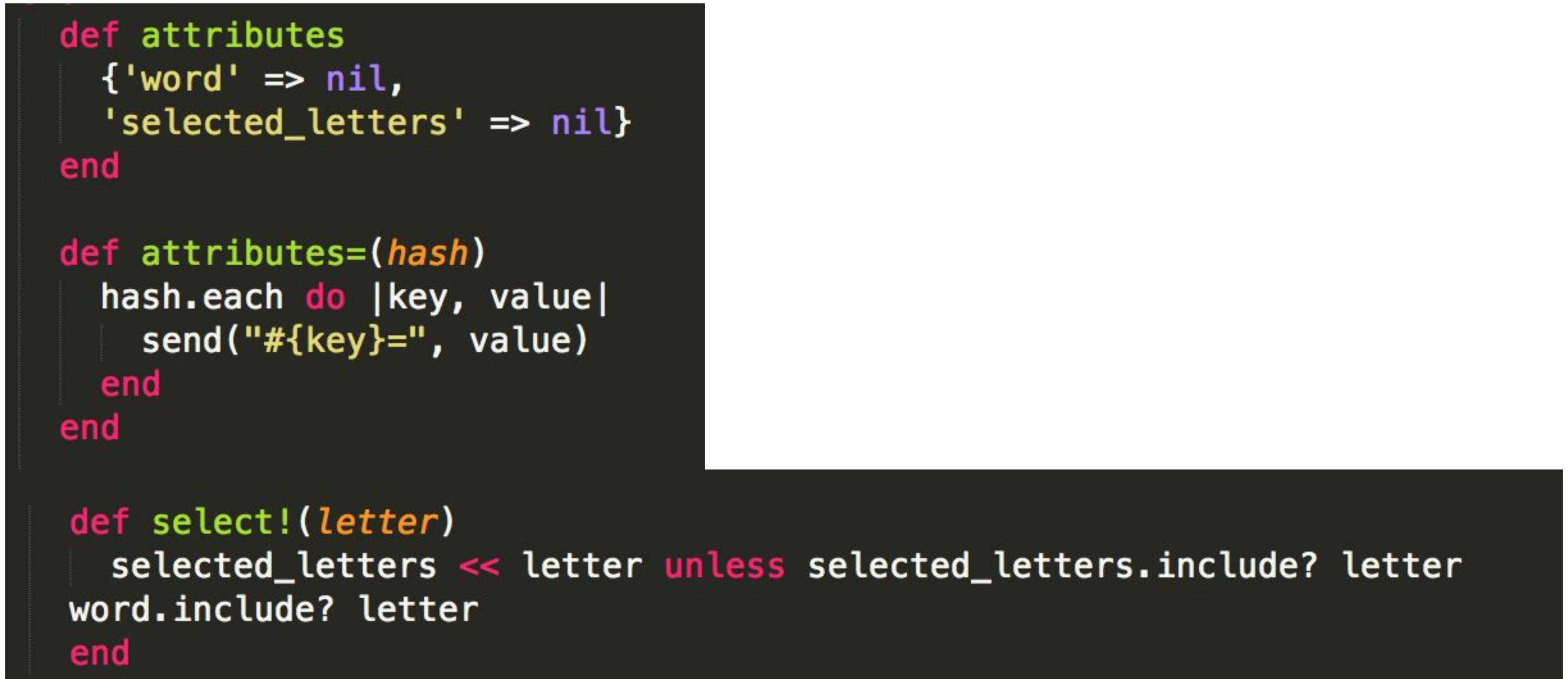
Challenges

- How to process user input during gameplay
- Solution
  - ● Game Controller:



```
def update
  current_game.select! params[:letter]
  update_current_game
```

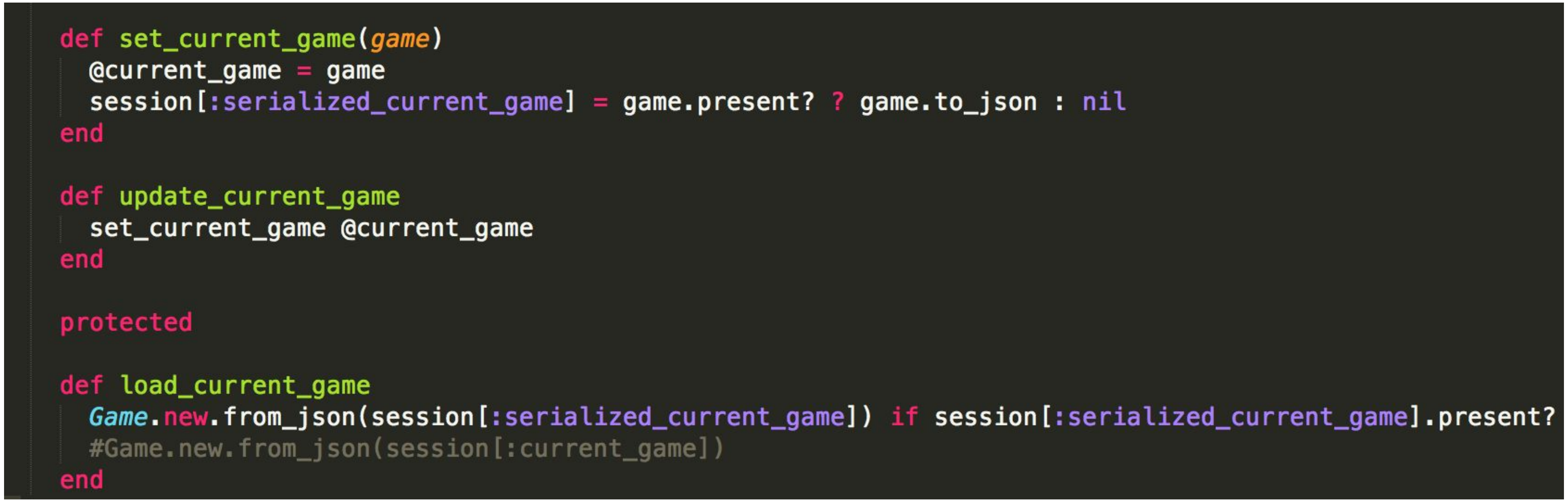
  - ● Game model:



```
def attributes
  {'word' => nil,
  'selected_letters' => nil}
end

def attributes=(hash)
  hash.each do |key, value|
    send("#{key}=", value)
  end
end

 def select!(letter)
   selected_letters << letter unless selected_letters.include? letter
 word.include? letter
 end
```

Challenges

- Using serialization to store game data in the session



```
def set_current_game(game)
  @current_game = game
  session[:serialized_current_game] = game.present? ? game.to_json : nil
end

def update_current_game
  set_current_game @current_game
end

protected

def load_current_game
  Game.new.from_json(session[:serialized_current_game]) if session[:serialized_current_game].present?
  #Game.new.from_json(session[:current_game])
end
```

If we had unlimited time…

- Create a visual hangman that adds a part each time you guess wrong
- Create more personalized accounts with discussion on leaderboard
- Get serializations to allow evil hangman to work properly