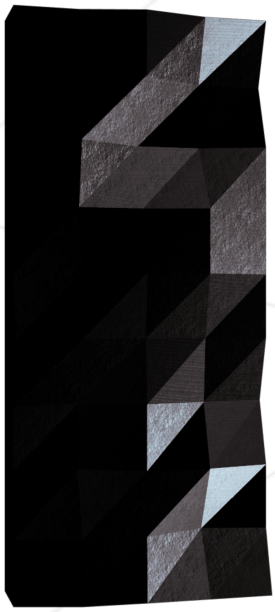

Ruby on Rails Training - Rush 00

Moviemon

Summary: *This is you first (relatively) complex project.*

Version:

Contents

Chapter I

Instructions

- Only this page will serve as reference. Do not trust rumors.
- The exercises have been ordered from easiest to most difficult. Under any circumstance you can submit or take into account an exercise if a previous one has failed.
- Be careful with the access rights of your files.
- You should follow the submit procedure for all you exercises.
- Your exercises will be corrected by your piscine peers.
- You cannot leave any extra file on your repository except the ones explicitly specify on you subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Everything you need can be found on the `man` or out there on Google.
- Read carefully the exercises: they may contain some features you should implement that are explicitly mentioned on the subject.
- Think about discussing on the forum Piscine of your Intra!
- Use your brain!!!

Chapter II

Today specific rules

- You must create a `stateless` application
- You must not use any `database`
- You must not use `ActiveRecord`
- This means:
  - Your class in the model(s) doesn't inherit `anything`
  - You controller(s) inherit ApplicationController (by default)
  - You can use (this is not a multi-user app) `4 global variables:` $view, $selected, $game et $player
- Save files will be stored in a valid JSON format.

Chapter III

Mandatory Part

III.1 Introduction - FAQ

This rush aims to make you code a solo game featuring a web interface.

`What's this game's goal?`
This game is named `Poke..` no, wait... This game is called `MovieMon`. Its goal is to capture all the... Moviemons hiding in a game grid, using, well, Movieballs.

`What is a moviemon?`
A `Moviemon` is a movie you can find on IMDb or themoviedb. Preferably, a monster movie.
.

`How do you catch a moviemon?`
Getting its life force to 0. Its life force will match the rating of a `Moviemon`'s film. A high rating will make the `Moviemon` harder to catch. The player's stength is equal to the number `Moviemons` they own. It increases their chance to catch other ones.

`How does a game go?`
When a player starts a new game, the game requests all the necessary films on IMDb and sends them on the 'Worldmap', the main page of the game.
The player will move freely over this page, square after square on a fixed grid. As they progress, they will harvest `movieballs`, or tumble over a `Moviemon`.
When they do, they must act: They can try to catch it. Or they can flee cowardly.
If their energy reaches down below 0 before the Moviemon, they flee and quit the game. Otherwise, the Moviemon is caught!
The player can then proudly consult their `Moviedex` that lists all the `Moviemons` they caught before returning to hunt, trying to catch them all! (Moviemons, that is, of course...)

III.2 Consignes

III.2.1 Rulez

Your game must observe a few rules:

- The front design must look like a grey portable game system with a green screen (see screenshots).
- Buttons must be clickable zones of the layout's main screen image-map
- The grid size must be at least 10x10 squares.
- The game must start with a screen title.
- The game takes place on the grid's screen showing a map image and a player image (showing their position).
- On the game's screen, 'start' button leads to "movie_dex". Another push leads back to the game's screen.
- If all functionalities via keys are present, they must be indicated with words (ex: "Press [A] to skip").
- Clicking on an unspecified button doesn't prompt anything (not even an error message).
- The "movies_mons" must be loaded at the beginning of the game.
- The "movie_dex" is a list of captured "movie_mons". Buttons on left and right will scroll the pages of this list radially: going left from the first entry of the list takes you to the last entry of this list.

Here are movie databases: OMDB, which is an unofficial API of IMDB, or The movie database.

III.2.2 Game data

During a game, you will have to keep data from one page to another. A classical website would use cookies or a session system on the server side. But we're not gonna deal with a classical website.

Your must store data in the global variables. You have four of them at your disposal: 'view', 'game', 'player' and 'selected' for the menus.

In your project, you must also create the logic required for the update and the use of a file that will contain the following informations:

- The player's position on the map.
- The names (or identifiers) of all the `Moviemons` in the `Moviedex`.
- Complete informations of all the `Moviemons` of the game, matching the ones in your chosen database.

III.2.3 Data management

You must also create one (or several) class(es) that will have to manage these game data. This (these) class(es) must at least contain one of the following methods:

- 'initialize'
- 'save': writes valid game data in a JSON.
- 'load': reads the game data from a file and use them to assign these data to the game's variables.
- 'get_movie': returns an array or hash containing all the details of the `Moviemon`'s name passed in parameters and necessary to the `Detail` page.

You can add as many methods and attributes to your class(es) as you see fit.

On the title screen, the 'select' button displays 3 classical 'save slots' and load a save. On the game screen (the map), the 'select' button also displays three 'save slots' and you can save the current game. On both those 'save' screens, you can navigate with top and bottom arrow keys.

III.2.4 Aesthetics

The game will naturally be displayed on your browser via `HTML` and `CSS`. Using `Javascript` is prohibited.

The game's display must be split in two visually very distinguishable parts:

- `The screen`: displays the current game. There is no interaction available zone. It must not include any link or form.

- `Controls`: located below or on both sides of the screen, they allow you to interact with the game and they're contextual. This means their behaviors change according to the game's state. This also means they're not always active. However, even though inactive, they must always remain `visible and still`.

  There must be nine 'buttons':

  - Four `directions`, like the ones on a game paddle's cross pad:
    Up, right, down, left.
  - A button `select`.
  - A button `start`.
  - A button `Power` (the little red led)
  - A button `A`
  - A button `B`

You cannot add/delete 'buttons'. You cannot display any information in this zone either, except for the 'buttons" names.
Beyond this minor distinction, the aesthetics won't matter much in the mandatory part of the subject.
Button's behaviors for each view are described in the following section.

III.2.5 Pages

You must create the pages/views/behaviors listed below. A 'button' not mentionned in a page is an inactive 'button'. Besides, if the destination is not specified for a control, this means it returns the same page, with a potential modification.

`TitleScreen`

- Description: title screen.
- Screen: must display the name of the game as well as 'Start - New Game' and 'Select-Load'.
- Controls:
  - `Start` accesses the `Game`
  - `Select` accesses `Save slots`
  - `Power` "reboots" the game, resets the global variables and returns to the `Title screen`.

`Worldmap`

- Description: world map, where the playable character moves and tracks down `Moviemons`.
- Screen: a grid which size is specified in the settings. The square the player is on must display a character representation (image, font, etc...).
- Contrls :
  - `Directions`: each direction must move the character one square in the same direction. The player cannont go beyond the map.
    Each move can potentially make the character stumble upon a `Moviemon`.
  - `start`: accesses the `Moviedex`.
  - `select`: accesses the `Save slots`.

`Refreshing this page must not modify the character's position on the map.`

`Battle`

- Screen: displays the poster and the name of the `Moviemon`, its director's name, its power and yours.
  If caught, there must be phrase like "You caught it" displayed to mark the occasion.
  If fleeing the fight, there must also be something like "You coward!" appearing.

- Controls :
  - `A`: Hit `moviemon`
    the player hits and subtracts the hit_point from the moviemon's power. the player is hit back with a hit_point matching the moviemon's rating.
    If successful, (moviemon's power reaches 0 before yours), the `Moviemon` is caught in the `MovieDex`. You will display a suitable message and set back the character to full level, adding one hit point (experience needs to be rewarding).
    If the player fails, you will display a suitable message and will delete the moviemon from the list of available moviemons. The player is sent back to the map.

  - `B`: Fleeing to the `Worldmap`
    If the moviemon is set free, the player's power is restored and they're sent back on the map.

`Moviedex`

- Screen: A captured `Moviemon` with its information:
  Year, genre, director, rating, abstract AND poster.
- Controls:
  - `Right and left`: directions allow the selection of different films. You must use at least two directions: left and right.
  - `start`: Returns to the `Worldmap` page.

`By default, the first film of the list is selected when arriving on the page.`

`Save slot`

- Description: Allows to save game's progression in the 'save.json' file or the to 'load' a save file from the title screen.
- Screen: displays the three save/load slots.
- Contrôles :
  - `up and down`: select the desired slot.
  - `Select`: returns to the previous screen.
  - the others: `AD LIB`

Chapter IV

Bonus part

Once your mandatory part achieved, you can implement additional functionalities to earn bonus points.

If your assessor considers these functionalities useful and operational, you will have to explain how you've implemented them.
An error you cannot manage will invalidate this functionality.

Javascript will be tolerated ONLY in the bonus part as long as it doesn't interfere with the working of the mandatory part (which must operate without any JavaScript). AJAX and WebSockets are not authorized.

Here are some ideas of things your could implement:

- Assign the controls to specific keys so you can play without the mouse.
- In order to bring variety to the game, have each game select a different set of monsters.
- Add variety to the `Worldmap` setting impassable elements.

Chapter V

Turn-in and peer-evaluation

You must turn-in perfectly configured `Rails` project.

Except for what's required in the subject, you're free to organize the project as you see fit.

So server error will be tolerated. Thoroughly test the behaviors of your views.

You must provide a requirement.txt file containing all the libraries useful to the proper functioning of your project.

You models mustn't inherit anything. Rails may generate migrations but nothing in the db

Easy check with the following command executed in the console:

```
ActiveRecord::Base.subclasses.map { |forbid| forbid.name }
```

Chapter VI

Turn-in example

Figure VI.1: Your titlescreen could look like that

Figure VI.2: This Moviemon game appears to happen in A familiar map

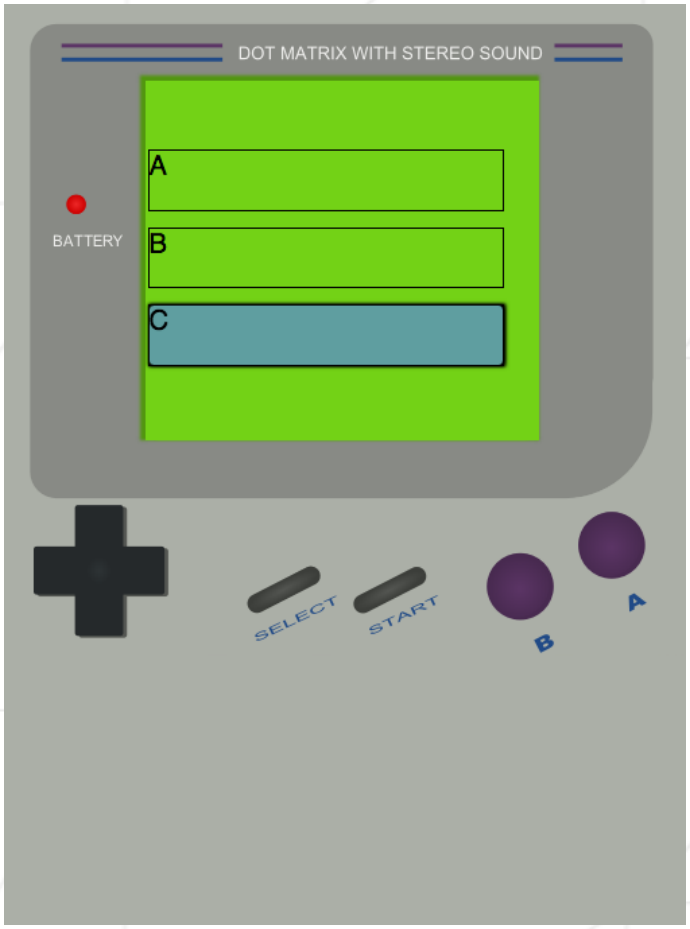

Figure VI.3: Save slot with 'c'selected

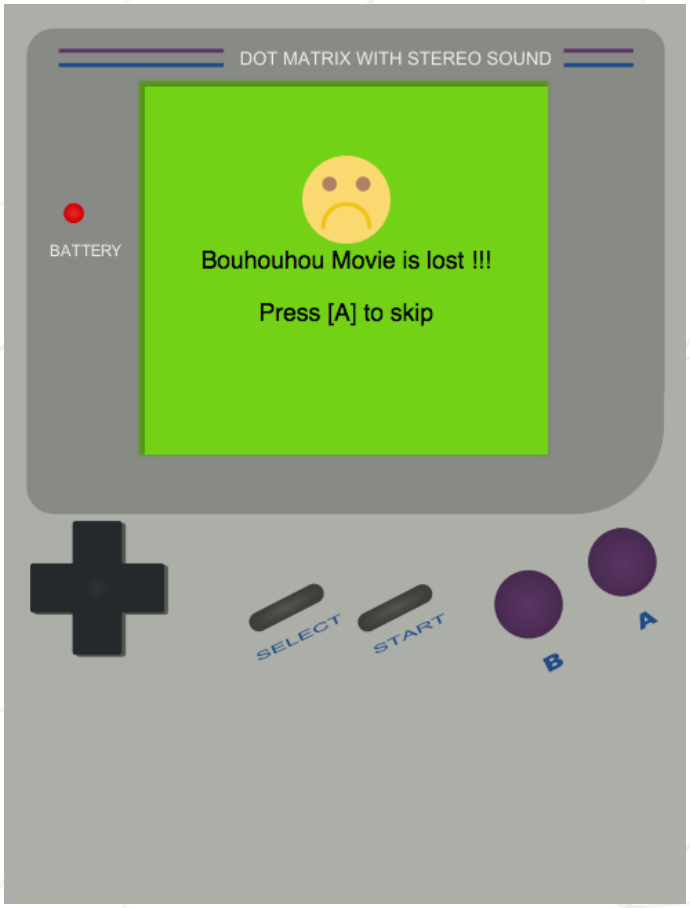

Figure VI.4: A lost combat message

Figure VI.5: A fight screen

Figure VI.6: Your moviedex could look like that