Frontend Basics

Practice in JavaScript and Html. Estimated Time: 3 days

2019

# General Requirements

These requirements apply to “Frontend basics” task as a whole.

1. One task is *exactly* one commit.
2. Attach the screenshot of working application to your merge request
3. Estimated time includes implementation of all tasks (3 working days for tasks).

Useful links:

<https://github.com/leonardomso/33-js-concepts#readme>

<https://learn.javascript.ru/>

<https://medium.com/@ajmeyghani/javascript-functions-a-pocket-reference-d42597ceb496>

<https://tylermcginnis.com/beginners-guide-to-javascript-prototype/>

<https://www.youtube.com/watch?v=JrmDTZPsla4&index=5&list=PLKaafC45L_SSA7_fLU7EMXTWI6ljhLx50>

1.    Game of “Life”

Create some kind of life simulation game. You have an environment where life exists. It is represented by two-dimensional array of cells. In this environment, there are predators and preys. Predators can eat neighbor preys or just move to a free cell. Preys can just move to free cells. If after some time predator has no meal - he dies. After some period, predators and preys can reproduce itself (take one free neighbor cell). Simulation ends when some amount of iterations is reached or when there are no life cells anymore.

Input data

1. Size of environment field.
2. Number of iterations.
3. Density of predators and preys.
4. Density of life cells.
5. Time of predators’ life and reproductive period.

Requirements

1. Make input data configurable via UI elements.
2. Visualize game process (environment field with cells) using one of the next ways. Find the most suitable for you:
3. HTML5 Canvas (<https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial>) (apparently the best way).
4. HTML DOM elements like table or div grids (also not bad).
5. Just write field matrix to DOM (far from being the best option).
6. Main goal of this task is to be acquainted with JavaScript language so feel free to use the latest features of ES6 syntax and try to implement the task using functional approach.
7. Do not mix the logic with representation; make visualization code independent from game logic code as much as possible.
8. Field should be generated completely randomly depending on life cells density (amount of non-zero fields), predators and preys density.
9. Neighbor cells are all cells around, i.e. top, left, right, bottom and diagonal.

Example

Input data: size - 2, life density - 0.5, predators’ density - 0.5.

0              1

0              -1

Here positive numbers will be predators, negative – preys; zero – free cells; life density equal to 0.5 means that half of the cells should be free and another half of the field cells should have predators or preys.

After one iteration, the field can look like this:

2              -1

0              0

Here predator moved from (0, 1) cell to (0, 0); prey moved to (0, 1); it was first iteration so predator became older what is reflected in his cell value (2); if we set predators life to 3 then on the next iteration it should die. Cell value will be set to zero.

After the next iteration, the field can look like this:

0              1

0              0

In this iteration, a predator from the cell (0, 0) ate neighbor prey on (0, 1) cell, and after this his age reset to 1.

After one more iteration, if we have configured a period of reproduction to be four we should have something like this:

0              2

1              0

Here a predator became older and reproduced itself into cell (1, 0).

Additional notes

* You can use an approach that differs from the one that used in the example.
* For visualization, just represent predator and prey as cells with different colors.
* Deal with the age in the code not in the visualization.
* Feel free to use your imagination and creativity in this task.

2.    15-puzzle (1) or Minesweeper (2)

Please implement just one of the games described in this section, depending on your *name length* in English in UPSA, i.e.

**Ivan --> 4 --> even --> Minesweeper**

15-puzzle *(odd)*

If you meet this game for the first time, here you can find the description: <https://en.wikipedia.org/wiki/15_puzzle>

Functional requirements

The game should include a header, a main field (4\*4 tiles) and a footer.

The header should display the following information: number of moves and a stopwatch. A stopwatch starts counting when you press “New Game” or “Restore game” button and stops counting when the game is over.

The footer should include two buttons: “New Game” and “Restore Game”. These buttons perform corresponding actions.

In case there are no available combinations for the user, the “Game over!” message should be shown.

If the user has successfully completed the game, the “You win!” message should be shown.

Technical requirements

1. ES6
2. Bootstrap

Minesweeper *(even)*

If you meet this game for the first time, you can find the description of this game here: <https://en.wikipedia.org/wiki/Microsoft_Minesweeper>

Functional requirements

The game should include a header, a main field and a footer. The headers should display the following information: the number of remaining bombs and a stopwatch. A stopwatch starts counting when you press a “New Game” button and stops counting when the game is over.

The footer should include a “New game” button and a “Number of bombs” input field. The user can set the number of bombs and the main field should be appropriately rendered. When the number of bombs increases then the number of cells in the main field increases. Default values are 8\*8 main field and 10 bombs.

If the user stumbles on a bomb, the “Game over!” message should be displayed.

If the user has successfully completed the game and indicated all bombs, the “You win!” message should be displayed.

Technical requirements

1. ES6
2. Bootstrap

3.    Calculator

Create a calculator application. A range of functions depends on you (+, -, /, \* and others). It should look like an ordinary calculator, where you have a bunch of buttons with the digits and operations, and a field where you can see the result.

Functional requirements

1. User should be able to make actions one by one using app UI elements (click digits and operations and see a result, e.g. 2.01 + 2.05 and user gets 4.06);
2. User should be able to do sequence of actions and then get the result (e.g. 2+3\*2-5 and then ‘=’ and get the result of this expression);
3. Keyboard should be supported.

Technical requirements

1. ES6
2. Bootstrap.
3. Cover functionality with unit tests *(Optional. Make it if there is enough time at the end)*.