UNIVERSITAT JAUME I

SJL 001 – Programming

Assignment 2 - Bejeweled

In assignment 1, you implemented a simplified version of the popular card game "Bejewled". For assignment 2, we are going to implement a web-based interface for our Bejewled game. The web-based interface needs to 1/ show us all relevant information related to the game (e.g. the amount of rows and columns for the game, the score, amount of turns left), 2/ allow us to interact with the game (e.g., select a cell). As novelty, now it needs to be possible to play as many games as we want, so there needs to be a way to start a new game, and we will play using 5 turns and minimum 6 colors.

You can re-use all code (abstractions, data, functionality) you wrote for the 1st assignment – but don't be afraid to update your code if needed!!

We will implement this assignment step by step, in 4 sub-assignments. Each sub-assignment incrementally builds towards the final application.

Assignment 2.1: generate the initial state of the game (25%)
In the first step of the assignment, we will implement only the functionality to generate the game. Make sure you have the following functionality:

- Make sure the user is able to input the amount of rows and columns for the game.
- The player can push a button to generate and visualize the game. In other words, when pressing the button, the game is injected in the HTML file with an initial setup: an X x Y game grid, with in each cell a random color.

No styling is needed, focus on the functionality.

Assignment 2.2: allowing the user to select a cell and mark adjacent cells (25%)
We now have the initial setup of the game ready. Now, let's add some interactivity to our game:

- The user must be able to select (i.e. click on) a cell.
- Once the cell is selected, we mark this cell, and we identify (you may for example mark them by given them another color, or changing their appearance) the adjacent cells (i.e., the cells north, east, south, west) that have the same color as the selected cell.

You may need to extend your solution of 2.1 so it's easier to implement assignment 2.2.

No styling is needed, focus on the functionality.

Assignment 2.3: implement falling down of colors (25%)
You are now able to select a cell and you've identified the cells that will be cleared. Now, we'll add the functionality of falling down of colors, according to the cells that were cleared. Consequently, we will add the following functionality:

- Moving down of colors into cleared cells (i.e., the colors "fall down")

No styling is needed, focus on the functionality.

Assignment 2.4: finalization (25%)
Now, we will finalize and "beautify" the game. Add the following functionality to your game:

- Visualize the score of the player, and update it according to the cells that were cleared.
- Visualize the amount of turn the player has left, and update the amount of turns left after every turn (i.e., the amount of turns is decremented, or stays the same if the player cleared more than 3 cells).
- Stop the game once the player has no turns left.

A basic version of the game could look as follows:

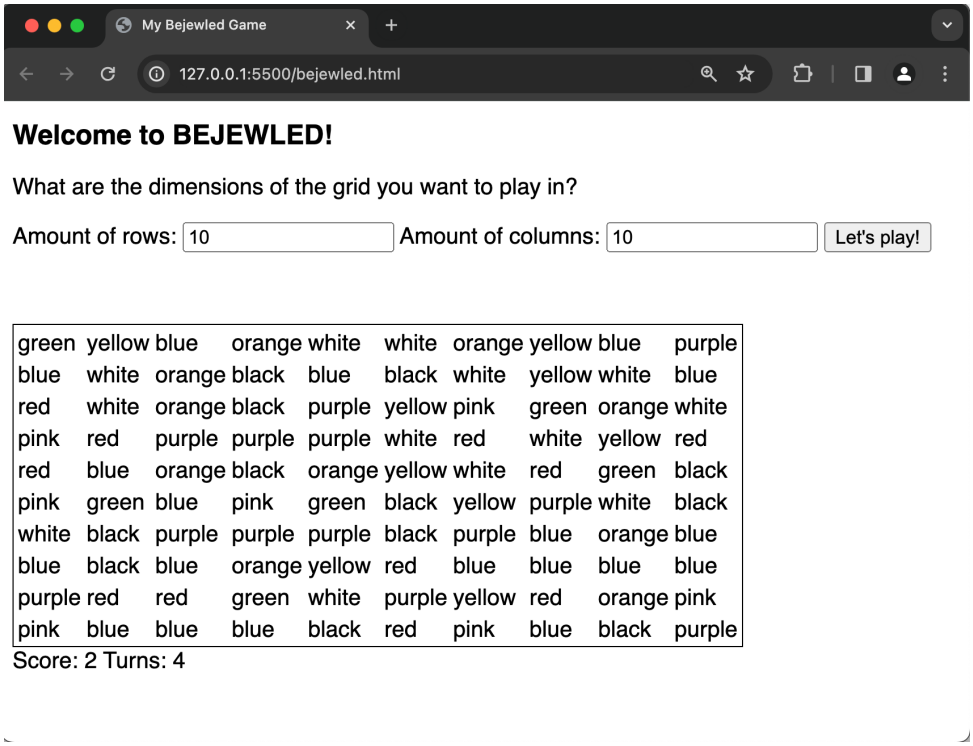

Figure 1 A basic version of the Bejewled game.

- Create a nice style for your webpage (using CSS)
- Improve the look and feel of your game

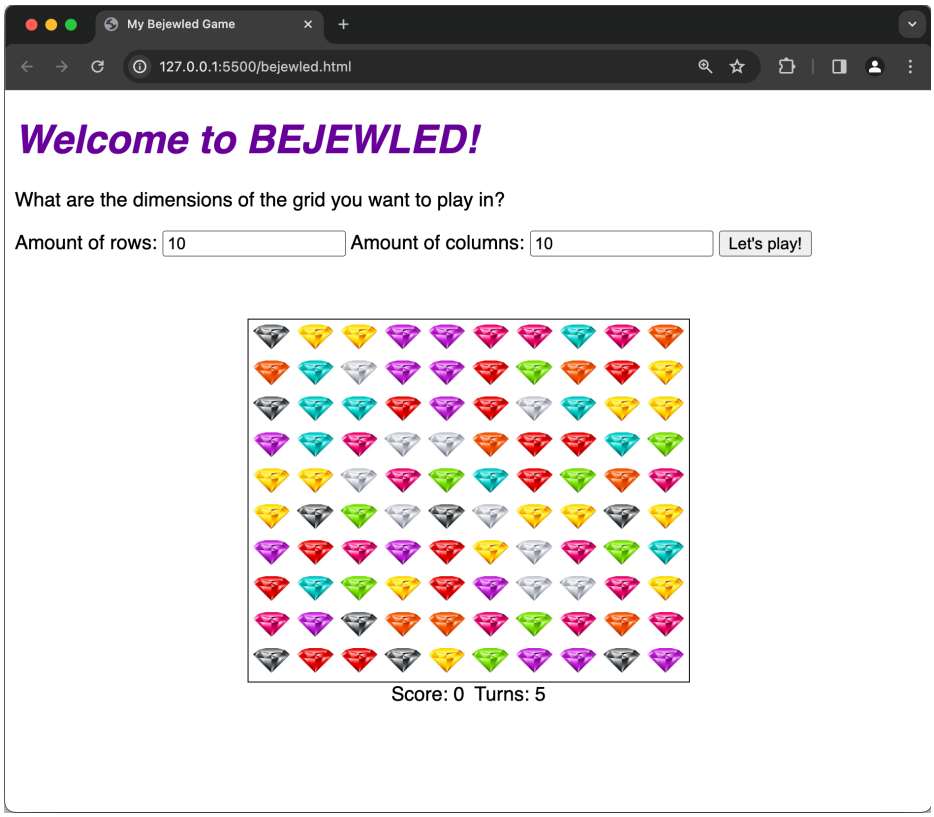

Figure 2 An example of a beautified version of the game.

Rules of the game (same as assignment 1)

The game consists of a n x n grid, and with m (min 6) colors. For example, the game can be played in a 6 x 6 grid, with 4 colors. Upon starting the game, the user is presented with a randomly generated n x n grid, where each cell contains one of the m colors, e.g., Figure 1.

Now, the user can choose (click on) any cell. As a result, the chosen cell, and all adjacent (horizontal and vertical) cells of the same color, are cleared. For example, in Figure 2, the user choses the cell circled in black; cells indicated with a green arrow are cleared, cells indicated with a red arrow are not cleared.

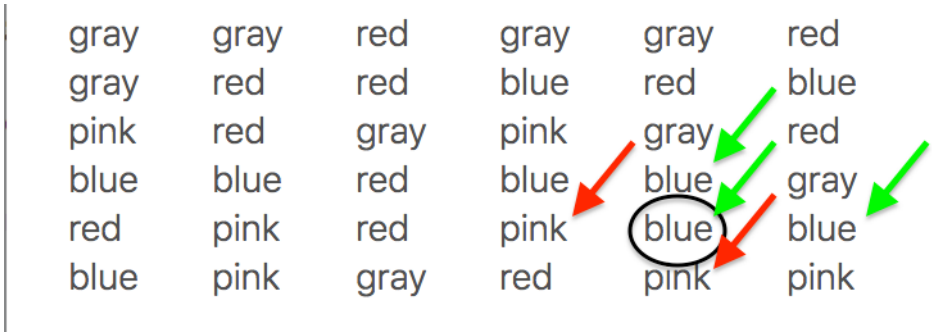

Figure 3 User selecting black circled cell; green arrowed cells are cleared.

For every cell cleared, the user gets a point. In the above example, the user scores 3 points: 1 (selected cell) + 2 (adjacent cells). If the user scores 4 or more points (in one turn), he receives a bonus turn (i.e., an extra turn)!

Subsequently, colors fall down vertically to fill the cleared cells, and new random colors fall down from up to fill resulting empty cells at the top of the grid. For example, Figure 3 shows the result of the previous turn (i.e., selecting the black circled cell in Figure 2): in the fifth column, “gray”, “red”, “gray” fell down, and “red” and “red” fell from up (new colors). In the sixth column, “gray”, “red”, “blue”, “red” fell down, and pink fell from up (new color). In the fourth column, nothing happened.

Figure 4 Result of turn; blue arrows represent falling colors.

The game finishes after n turns (+ any bonus turns the user earned), and the final score of the user is the total amount of cells he could clear.

Tips

- Don't panic! We can do this!
- Think about the solution before you start implementing!
- You have the basic functionality of the game already implemented (assignment 1). You can re-use all code (abstractions, data, functionality) you wrote for the 1st assignment – but don't be afraid to update your code if needed!!
- Use **separation of the concerns**!
    - The HTML file will contain the structure and content to be visualized to the user. Prepare your HTML file so you can easily access the HTML elements you will need to manipulate (i.e., give them IDs, class names, … whatever is needed). You can fully prepare your HTML file to contain the structure and most information (i.e., explanatory text, input fields, button to generate the game), but some content will need to be injected from the JavaScript file (i.e., the grid corresponding with the game, according to the dimensions entered by the user).

      You may add some event listeners and the corresponding event handlers (i.e., actions – calls to JavaScript functions) directly in HTML (for example, for the button to generate the game), but you will also need to add (some – at least one) event listeners and attach the corresponding action in the JavaScript file).

    - The CSS file will contain all styling information. This defines the look and feel of your HTML page. Preferably, don't add style information directly into your HTML file.

    - The JavaScript file contains the functionality. Here, you will
        1. Store, keep track of and update all the necessary data (e.g., the score, amount of turns left, colors, …).
        2. Define the functionality (e.g., generating the game grid, identify cells to clear, clearing cells, falling down of colors, …).
        3. Implement the event handlers (= JavaScript functions), which are activated ("triggered") when events are raised in the browser (e.g., pressing the "Let's play" button, pressing a cell).
        4. Interact with the HTML page (using the DOM) to visualize the relevant data generated during the game (e.g., the game grid, updating the game grid according to cleared

cells and colors falling down, the score, the amount of turns left, etc.)

- Remember, **divide and conquer**!
  - Create the game step by step. Follow the sub-assignments – they gradually define the functionality of the game and allow you to implement one functionality at a time.
  - Test every function along the way; make sure they work before continuing.

- For each functionality to add, generally follow the following steps (note that sometimes, not all steps are needed):
  - Prepare the HTML file:
    1. Add the necessary user interface element(s) to trigger the functionality (e.g., to generate the game grid, add a button "Let's play!" in the HTML file) – if needed. Add the necessary event listener (= event you will listen for, for example, "click") and associated actions (= the event handler, a JavaScript function that will be called).
    2. Add the necessary HTML element(s) so you will easily be able to visualize the result of the functionality – if needed. For example, a DIV element in which you will visualize the game grid.
  - Implement the necessary JavaScript functions
    1. Functions that implement the functionality (e.g., identify cells to clear, falling down of cells, update score, update amount of turns left, …). You should already have these functions from assignment 1 (but perhaps, you need to update them).
    2. Function(s) to visualize results in the HTML file (e.g., visualize the game grid, update/set the content of a cell (i.e. a color), show the score and turns left). To do so, find the element in the HTML file to write to (using the DOM), then manipulate the element to show the result(s) (again, using the DOM). For example, find the HTML element (for example a DIV or SPAN element) where you will display the score (for example, based on its ID), then update its content with a new score (for example, by updating the innerHTML property).
    3. Event handlers: functions that will be triggered when an event happens in the browser (for example, the "Let's play!" button was pressed, a cell was selected). In such an event handler, you call the function(s) that implement the corresponding functionality, and if needed (in most cases), the function(s) that visualize the result.

Hints

To help you, consider the following hints (depending on your solution, you may not need all hints):

- We can use images with different colors to represent the colors. If you do so, make sure you use good naming for your image files, so you can easily retrieve them from within your JavaScript file.
- To change an image (= replace one image by another), we can simply change the "src" property of the HTML image element. The corresponding new image will automatically be displayed.

Extras (beyond the assignment):

- A visual "falling down" effect is not easy to create. It uses timeouts, which is something we haven't seen in class. So, not mandatory for the assignment (i.e., you can achieve a 10/10 with a perfect solution without the "falling down" effect)!!

  But if you do feel brave enough, here is how to use a timeout (i.e., myFunction() will be executed after a timeout of 500 miliseconds):

```
let timeOut = 500;
setTimeout(function(){
    myFunction();
}, timeOut);
```

  Be aware though, that the rest of your code continues to run – only the function wrapped in a timeout will be delayed). In other words, if you use multiple timeouts (as is needed when multiple colors fall down), you will need to synchronize your timeouts (i.e., the first timeout should for example delay 500 miliseconds, the second should delay 1000 miliseconds, the third 1500 miliseconds, etc.)!

- Playing a single sound is easy. Here is how to do it:

```
let soundEffect = new Audio("/audio/mysound.mp3");
soundEffect.play();
```

  However, when playing multiple sounds, you will need to synchronise them, and thus you will need timeouts (see previous) – else, your sounds will all play virtually at the same time. So, not mandatory for the assignment (i.e., you can achieve a 10/10 with a perfect solution without any sounds)!!

Evaluation

- The evaluation takes into account three things: 1/ did you implement all the functionality correctly? 2/ did you produce a *good* solution (e.g., good programming style, separation of concerns, short functions, etc.)? 3/ does your page and your game has a nice look and feel (i.e., appropriate use of CSS for a basic style, any other visual improvement such a use of colors or images to present the colors in the game)?

- Even if you don’t manage to implement all functionality, you can still get a passing or good grade. On the other hand, implementing all functionality doesn’t guarantee a maximum score.

- Handing in late (until 1 week) will result in -20% score for this assignment. Handing in more than 1 week late will result in a 0 (“failed”) for this assignment.