CSE104 project

Author: Long Vu(2048 Solitaire) + Tung Nguyen(2048 + main interface)

Table of content

Introduction

This client-side website is inspired by the very famous game `2048`. The `2048 series` contains two games (maybe we will add more game later):

- The original `2048`.
- `2048 Solitaire` - a combination of `2048` and another famous game - `Solitaire`.

The original `2048` game

Game state

The game state is defined by a 4x4 matrix together with the highest score and the current score. This state can be store in the local storage so that if the players reload the website, they can still play the previous game.

Merging

For merging tiles, we only implemented merging along the left direction. To get merging along the right, up, and down direction, we use the reverse/transpose of the board, merge along the left direction and change it back to the original state.

During every merging event, we changed the state of the board inside the JavaScript code first. After finish changing the state of the board, we update the HTML to match the board and store the new board to the local storage.

Endgame

In the beginning, we set the winning state to `display: none` to make it hidden.

After every merging event, we check whether the game ends or not, that is the player got the `2048` tile or they cannot make a merge action. If the game is ended, we will show `Game over` or `Victory` together with the `Try again` button by changing their `display`.

How to play board

We created a `How to play` button. Whenever we hover on the button, the board will slowly disappear and the instruction will be shown. We used `requestAnimationFrame` to make the board slowly disappears by decreasing the CSS property `opacity` of the board. After the board disappears, we show the instruction.

Show keys option

To make the game available with only a mouse or for playing with a phone, we added arrow keys to the game. You can show or hide the key by pressing the `Show keys` button. The key is one arrow image and its rotation.

`2048 Solitaire`

Game state

We represent the card using `div` and with the number inside using `p`. 4 columns are corresponding to 4 blocks, inside each column, there's one div who is the bonus indicator who will appear later, another div that marks the bottom of a column for adding a card. Each card in the column has the distance of `30px` in their `top` attributes. Each time adding a card to a column, we append the `div` card into each `div` column. We have a `New game` button that resets the game state by erasing the columns and a `How to play` button that shows the instruction. We have 2 cards at the bottom, id `one` and id `two`, each time a card is placed, we push id `one` to id `two` and generate a new random card for id `one`. We use the `drag and drop API` (https://www.javascripttutorial.net/sample/webapis/drag-n-drop-basics/) of `JavaScript` to implement the dragging and dropping effect of the cards. In each column, we have a class `cur` that indicates the current bottom card that can be added to, so a white border will appear when we hover the card over the last card and not in the others, and we are only able to add cards in the last card. We also add 4 `drag and drop` listeners to the current last card and the card that we dragged. We also make sure that we only drop on the card but not the number of the card(https://javascript.info/bubbling-and-capturing).

Most of the `HTML` and `CSS` we used are from the course, the HTML structure is clearly defined in the code. We divide the HTML tree into 3 parts, `top`, `middle`, and `bottom`. The top contains the max score, the current score, the `how to play` button, and the `new game` button. The middle contains the 4

columns and the cards that are going to be added later, a dotted line indicates the maximum number of cards a column can have. The `bottom` contains 2 cards that are continuously generated when we play. We used fixed `height` and `width` for some parts, also fixed `top` and `left` pixel values to align the cards properly. Most of the `CSS` is used for aesthetic purposes, alignments and is common in the `HTML` tutorial.

We also use `localStorage` to keep track of the maximum score.

Merging cards

Each time a new card is placed in a column, we check 2 bottom cards if they have the same numbers, and we merge them by deleting the last `div` and changing the number of the penultimate `div` . We use `requestAnimationFrame` to visualize the merging process, moving the `top` attribute of the last card by a marginal amount of `px` until it reaches the penultimate card if at the end of the process 2 bottom cards still have the same number we call `requestAnimationFrame` again(`recursive` style). Also, we keep count of `bonus` , which is how many time the merging process has taken place, and we have a small box indicating the bonus amount(`2x` , `3x` ,...) with the `top` attribute `=` the `top` attribute of the penultimate card `+40px` .

Endgame

We keep track of the `maxCard` variable representing the maximum `log2` of cards on the board, we update the `maxCard` when the cards are merged. After a merging process, if `maxCard` is bigger than 11, it means that we have `2048` on the board and we disable the dragging effect of the card and changing the `display` of the winning screen and the columns have `display` none. The same logic with checking the losing state except the conditions need to be more careful(if all columns are full and no last cards of the columns match the current card).