# Advance Web Development 2 – Prelim Project: Memory Game

## Introduction:

This is a mechanics on how to build a memory game using Javascript. The idea of the game is try to match two cards that are the same. And if they don't match you move on to flip other cards and remember their position. The game ends when the whole board is uncovered. Is a very simple game.

## Instructions:

1. Create a project folder and add the provided files (index.html, lodash.min.js)

2. Then create empty files in that project with the name: style.css and actions.js.

In the style.css add the corresponding style.

```
div#memory_board {
 background:#DADADA;
 border:#999 1px solid;
 width: 800px;
 height: 540px;
 padding: 24px;
 margin: 0px auto;
}
```

Inside the memory board we will place a lot of divs, that will be the cards. They will be placed dynamically using Javascript. But before that, we need to add one more style to the style.css to represent the back of the card.

```
div#memory_board > div {
 background: #FF3399;
 border: #000 1px solid;
 width: 71px;
 height: 71px;
 float: left;
 margin: 10px;
 padding: 20px;
 font-size: 64px;
 cursor: pointer;
 text-align: center;
}
```

3. Then we open the actions.js and we will start by creating some variables.

```
var memory_array =
['A','A','B','B','C','C','D','D','E','E','F','F','G','G','H','H','I',
'I','J','J','K','K','L','L'];
 var memory_values = [];
 var memory_tile_ids = [];
 var tiles_flipped = 0;
```

The variable memory_array is the collection of the content of the cards, this array will be used to create the cards dynamically. In this case it will be pairs of letters. But we can change that to anything we want.

The next variables will be populated during the game play script, we will come back to their usage later.

4. The next step is to create the newBoard() method. This method will be executed everytime the game starts. This function will generate all the cards in the board as little divs.

```
function newBoard() {
 tiles_flipped = 0;
 memory_array = _.shuffle(memory_array);
 var output = '';
 _.forEach(memory_array, function(memory_array_value, index) {
 output += '<div id="tile_'+ index +'" onclick="memoryFlipTile(this,
\''+ memory_array_value +'\')"></div>';
 });
 document.getElementById('memory_board').innerHTML = output;
}
```

This function first will set the tiles_flipped variable to zero as we are starting a new game. Then it will shuffle the memory_array as we want to have the cards in different position in every game. For shuffling we will use the _.shuffle function that lodash provides, so we don't need to build our own. It makes the code much cleaner and easy to mantain.

Then we will create a new variable called output that is were we will be adding our just created divs. And we will populate that output variable iterating over all the values in the memory_array. For iterating over all the values we will use the _.forEach function in lodash. In every iteration a new div will be created and add it to the output variable. The div will contain the index of the memory_array value and action to execute when the card is clicked.

At the end we just assigned the output variable (that contains all the card divs) to innerHTML of the memory_board.

Now every time the pages loads the newBoard() function will be called. We can open index.html in the browser and we can see that the board is full of cards now.

5. So when we click in a card memoryFlipTile() function will execute. This function takes as a parameter the div where the function is called from and the value of the memory_array array in the position of that card.

To build the memoryFlipTile function, we need to think of the different stages of the game. So we can start by creating the function thinking about what will happen in each case. For this stage we just write the logic of the game in an structured form. The end result is following and it is written valid Javascript but it is also quite clear to read in English.

```
function memoryFlipTile(tile, value) {
 if (canFlipCard(tile)) {
   flipCard(tile, value);
   if (areNoCardsFlipped()) {
     setCardAsFlipped(tile, value);
   } else if(isOneCardFlipped()) {
     setCardAsFlipped(tile, value);
       if(isThereIsAMatch()) {
         matchCards();
         if (isGameOver()) {
           gameIsOver();
         }
       } else {
         cardsDoNotMatch();
       }
     }
   }
 }
```

After we have created the logic for our game, we need to start populating those functions. And those will be really simple functions, as we already have cracked the problem into so small parts.

6. *canFlipCard(tile)*

So everything starts with the tile (the card div) and its content (the letter value of the card in this case). The code will only execute if this statement is true.

*if(tile.innerHTML ==”” && memory_values.length < 2)*

This means that there is nothing inside the div and that the amount of memory cards turned is less than two (so this will only execute when there is no card selected or when there is one card selected).

```
function canFlipCard(tile) {
 return tile.innerHTML == "" && memory_values.length < 2;
}
```

7. *flipCard(tile, value);*

The card can be flipped, so let’s flip the card, and that means changing the style of the card so it looks that it was flipped, we set the background color to white and the innerHTML to the letter value. So when we click the card we see the letter value on a white background.

```
function flipCard(tile, value) {
 tile.style.background = '#FFF';
 tile.innerHTML = value;
}
```

8. *areNoCardsFlipped()* and *isOneCardFlipped()*

We check that there are no cards already flipped or that there is only one card flipped, just by checking the length of the memory_values, as there is were will be storing the flipped cards.

```
function areNoCardsFlipped() {
 return memory_values.length == 0;
}

function isOneCardFlipped() {
 return memory_values.length == 1
}
```

9. *setCardAsFlipped(tile, value);*

To set the card as flipped we just need to save them in the memory_values. Also for practical reasons we will stored the identificator of the tile in the memory_tile_ids.

```
if(memory_values.length == 0){
 memory_values.push(val);
 memory_tile_ids.push(tile.id);
```

10. *isThereIsAMatch()*

Just check if the values of the two cards are the same.

```
function isThereIsAMatch() {
 return memory_values[0] == memory_values[1];
}
```

11. *matchCards();*

If the cards match then we can add 2 to the tiles_flipped variable. This variable will be counting all the tiles that were flipped so we know when the game is over. And we clear both arrays memory_values and memory_tile_ids as now those cards are a match.

```
function matchCards() {
 tiles_flipped += 2;
 memory_values = [];
 memory_tile_ids = [];
}
```

12. *isGameOver()*

If the amount on the tiles_flipped is the same as the length of the memory_array then we just completed the game and the board is cleared.

```
function isGameOver() {
 return tiles_flipped == memory_array.length;
}
```

13. *gameIsOver()*

When the game is over we send a message to the user telling that the game is completed, clean the memory_board HTML element and start a newBoard().

```
function gameIsOver() {
 alert("Board cleared... generating new board");
 document.getElementById('memory_board').innerHTML = "";
 newBoard();
}
```

14. *cardsDoNotMatch();*

If the two cards are not the same then after some time the cards will be flip back again. To achieve this we use a timeout set to 700 milliseconds, before turning the card back.

```
function cardsDoNotMatch() {
 setTimeout(flipCardBack, 700);
}
```

For turning the card back we create the function flipCardBack(). That will set the styles of each of the flipped tiles back to its original color and set the value to empty. At the end it will clear the memory_values and memory_tiles_ids.

```
function flipCardBack() {
 var tile_1 = document.getElementById(memory_tile_ids[0]);
 var tile_2 = document.getElementById(memory_tile_ids[1]);
 tile_1.style.background = '#FF3399';
 tile_1.innerHTML = "";
 tile_2.style.background = '#FF3399';
 tile_2.innerHTML = "";
 memory_values = [];
 memory_tile_ids = [];
}
```

15. 17. Now the game is completed and you can try it in your browser.