# COMP 2537 Web Development 2

# Assignment 3

Introduction:

The objective of this assignment is to design and develop a memory card game using HTML, CSS and JS.

The memory card game is a matching game where all cards are laid face down, and players take turns flipping over two at a time, trying to find pairs with the same image. The goal is to remember where cards are and collect the most matching pairs.


(https://en.wikipedia.org/wiki/File:International_Pok%C3%A9mon_logo.svg
https://www.businessinsider.com/most-popular-pokemon-fan-vote-reddit-survey-2019-6)

The game will be Pokémon themed and use the Pokiapi.co API server to get images for the front of the cards that correspond to various Pokémon.

Instructions:

Create a card game using Pokémon cards. In this game, as a player, you'll be playing against the timer to see if you can match all the cards before the time runs out.

For each new game, retrieve a different set of Pokémon randomly from the Pokiapi.co API server. No two games will have the same cards (at least be very statistically improbable). Ensure that each Pokémon can only be assigned to a single set of cards and cannot be used for multiple cards.
For example, if Charizard is picked, only 2 cards will have Charizard, but no other set will contain Charizard.

You will create different game difficulty levels: easy, medium, and hard. Adjust the number of cards and time limits according to the chosen difficulty. The game will end when either the player has matched all the Pokemon sets (show a "winning" message and prevent the cards from flipping) or the timer runs out (show a "game over" message and prevent the cards from flipping). Players should be able to restart the game by clicking on the "reset" button or by changing the difficulty level and clicking "start".

Display a status header showing the number of clicks the user has made, the number of pairs left, number of pairs matched, and total number of pairs.

Allow users to select a theme. For example, a user can choose between dark and light themes.

Add a power up feature. For example, allows the player to see all the cards for short time. How the power up feature is triggered is up to you.

User Stories:

As you know, user stories are a key part of the agile development methodology as they help define valuable software features from the perspective of the end-user.

Here are some examples of user stories being implemented in this assignment:

- As a *game developer*, I want to retrieve Pokémon randomly from the Pokiapi.co API server, so that **each game session offers a unique experience** set of Pokémon cards.

- As a *player*, I want to play a card game utilizing Pokémon cards, so that I can enjoy the game with **familiar and exciting Pokémon**.
- As a *player*, I want to have **different difficulty levels** (easy, medium, and hard), so that I can choose the level of challenge that suits my skill and preference.
- As a *player*, I want the number of cards and time limits to adjust based on the chosen difficulty level, so that **the game remains balanced** and appropriately challenging.
- As a *player*, I want to ensure that each Pokémon can only be assigned to a single card pairing, so that **the game follows the rules of memory card games** and offers a fair gameplay experience.
- As a *player*, I want **the game to have a start button**, so that I can initiate a new game session whenever I'm ready.
- As a *player*, I want **the game to have a reset button**, so that I can restart the current game session if I make a mistake or want to start over.
- As a *player*, I want to see a header displaying **relevant game information**, such as the number of clicks I have made, the number of pairs left, the number of pairs matched, and the total number of pairs, so that I can track my progress and stay informed during the game.
- As a *player*, I want to be able to select **different themes** (e.g., dark and light), so that I can customize the visual appearance of the game according to my preference.
- As a *player*, I want **a power-up feature** that helps me during the game, such as allowing me to see all the cards for a short period of time, so that I can strategically plan my moves and improve my chances of finding matching pairs.

Pokemon API:

The Pokémon API available at Pokiapi.co has an extensive collection of Pokémon and their stats.

To get a list of all the Pokémon available, you can go to this API endpoint:

**Pokemon List API Endpoint:**
https://pokeapi.co/api/v2/pokemon?limit=1500

In the above URL, I have provided a limit of 1500 so that we can see all the Pokémon. By default, the API will limit the result to just 10. As right now, there are just over 1300 Pokémon.

This endpoint returns a JSON object in the following format:

```
{
    "count": 1302,
    "next": null,
    "previous": null,
    "results": [
        {
          "name": "bulbasaur",
          "url": "https://pokeapi.co/api/v2/pokemon/1/"
        },
        {
          "name": "ivysaur",
          "url": "https://pokeapi.co/api/v2/pokemon/2/"
        },
        ...
    ]
}
```

It will give you the total number of Pokémon in the `count` property and will list all of the Pokémon in the `results` property. The `results` will be an array of objects, one for each Pokémon. Each Pokémon has a name and another URL to get more detailed information about that Pokémon.

Note that not all the Pokemon IDs are sequential. There are some gaps in the Pokemon IDs once you get beyond ID 1025.

To get more detail about and to get an image for a Pokemon you can go to this endpoint:

**Pokemon Detail API Endpoint:**
https://pokeapi.co/api/v2/pokemon/**<ID>**
(where **<ID>** is the ID of the Pokemon you are trying to get more details about)

ex: https://pokeapi.co/api/v2/pokemon/1 is for Bulbasaur 

and: https://pokeapi.co/api/v2/pokemon/25/ is for Pikachu 

This endpoint returns a JSON object in the following format (although some of the fields are optional and some Pokémon won't have all these fields):

```
{
  "abilities": [],
  "base_experience": 64,
  "cries": { },
  "forms": [],
  "game_indices": [],
  "height": 7,
  "held_items": [],
  "id": 1,
  "is_default": true,
  "location_area_encounters":
   "https://pokeapi.co/api/v2/pokemon/1/encounters",
  "moves": [  ],
  "name": "bulbasaur",
  "order": 1,
  "past_abilities": [],
  "past_types": [],
  "species": { },
  "sprites": {
    ...
    "other": {
      ...
      "official-artwork": {
        "front_default":
"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon
/other/official-artwork/1.png",
        "front_shiny":
 "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon
 /other/official-artwork/shiny/1.png"
      }
    }
    ...
  },
  "stats": [],
  "types": [],
  "weight": 69
}
```

(I have omitted some of the field values - ex: `abilities` and `moves` for brevity)

Most Pokémon (with IDs 1025 and below) will have a `sprites` field with an `official-artwork` field under `other` which will contain a `front_default`. This probably the best place to get an image for each Pokémon. There might be some other images that you come across in the `sprites` field, however, some of them are very low resolution (< 96 x 96 pixels) or the images may not always be in the same place. Some Pokémon will have a `generation-i` field with an image, but not all will.

Suggested Development Steps:

1. Set up the files and the imports.
2. Create 6 cards. Add two images to each card - front and back. Have the cards inside a flex box.
3. Overlap the front and back of each card so that only the backs of the cards are visible.
4. Add a flip animation to the cards on a mouse click event.
    a. Add the flip class to `rotateY(180deg)` to the `.card:hover` and transition in 1s
    b. Make the rotation in 3d by adding `perspective: 1000px;` to the `.card`
5. Check if the last two cards are the same.
    a. If the two cards are not the same, flip them back with some delay
    b. If the two cards are the same, remove them from the game (clicking on them again should have no effect).
6. Edge case 1: If the user clicks on the same card twice (i.e. it is already flipped over), do nothing.
7. Edge case 2: If the user clicks on a card while it is flipping (during the transition animation), do nothing.
8. Add the winning event. If the user matches all the cards (before the timer runs out), display a winning message.
9. Add a header showing the number of clicks the user has made, and the number of pairs left, number of pairs matched, and total number of pairs.
10. Add a timer to the game.
    a. Show the time in the status header.
    b. Add a game over message for when the timer runs out there are still unmatched cards.
11. Add a *reset* button to the game.
12. Add a *start* button to the game.
13. Add a difficulty level to the game. Show the levels controls to the header.
    a. Add the logic to the difficulty levels. Adjust the number of matches required to win and the time allowed.
14. Add themes.
15. Add power-up logic.

The starter code included in the assignment will help with completing steps 1 through 4.

Website appearance:

Part of your grade will be based on the visual design and appearance of your website. To earn full marks, make sure to enhance your HTML with thoughtful CSS styling that improves the visual design and user experience. Take the time to make your website look professional. You may use any CSS framework and/or code the CSS yourself.

Suggestions:

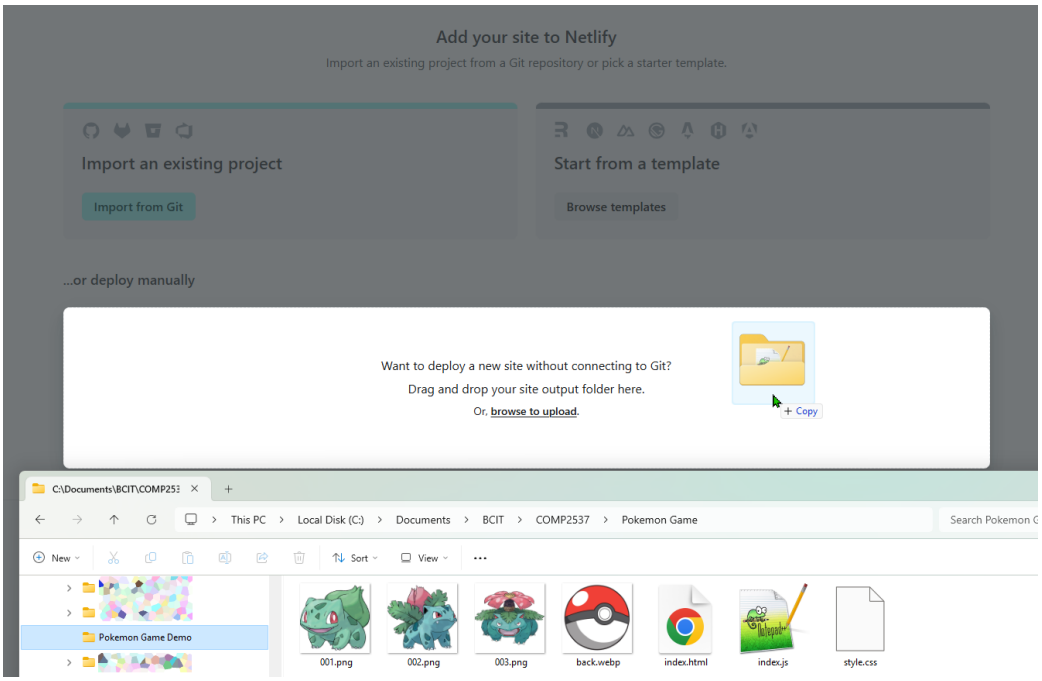
- Choose a clean, attractive color palette that suits the theme of the game.
- Choose a font and size that is appropriate for a Memory Game that has Pokémon cards.
- Style the cards to look like cards.
- Choose a style for buttons for starting and resetting the game. Use a consistent style for applicable buttons.
- Using spacing and alignment to give a consistent look.

Hosting on Netlify (or other static hosting service):

Create an account with Netlify (or another static hosting service).

I suggest Netlify for this assignment (and not Render) since this website doesn't require any server-side code. There is no database to connect to and no server-side JS. All of the code is HTML, CSS and client-side JS. Netlify is a great, very simple site to host static, client-side only code.

To create your Netlify site, all you need to do is drag your project folder into your browser once you have an account with Netlify:



Marking Guide:

| **Criteria** | **Marks** |
|---|---|
| Each game played, a new set of randomly determined Pokemon cards | 10 marks |
| Cards flip when clicked. | 2 marks |
| If the user clicks on the same card twice, do nothing. | 2 marks |
| If the user clicks on a card that is already matched, do nothing. | 2 marks |
| If the user clicks on a card while two cards are already flipped or are currently flipping, do nothing. | 2 marks |
| Winning Event is implemented: If the user clicks on all the cards, display a winning message. Losing Event is implemented: If the timer times out before all matches are found, display a losing message. | 7 marks |
| Status section shows the number of clicks the user has made, and the number of pairs left, number of pairs matched, total number of pairs, and game timer. | 7 marks |
| Start and Reset buttons are added to the game. | 7 marks |
| Difficulty levels are added to the game. | 7 marks |
| Themes are added to the game. | 7 marks |
| Power-up logic is added to the game. | 7 marks |
| The website has an appealing, professional and consistent look. | 10 marks |
| Note: You will receive a **mark of 0** if your code doesn't run<br>(i.e. too many errors to run properly, cannot connect to the database, etc.)!<br>I will be running the code on your Render app, please make sure it is available until after you receive your marks! | |
| **Total:** | **70 marks** |

Submission Requirements:

| **Submission:** |
|---|
| A link to your Youtube video demo.<br>ex: `https://youtu.be/c8nPuL5phG1` |
| A link to your GitHub Repo with your website.<br>(REMEMBER: Commit **AND PUSH** your code to GitHub!)<br>ex: `https://github.com/fluffybunny72/assignment3-studentname`<br>Include your self-graded checklist within the root of your git repo. |
| A link to your hosted site (on Netlify).<br>ex: `https://67f9ce7db83f6b967f0-teal-kheer-b24fd6.netlify.app/` |

Self-graded Checklist:

Find the self-graded assignment 3 checklist template and for each item, either **leave it blank** (if you did not complete the task fully) or put an **x** next to it (indicating you completed the task).
Provide a totalled grade out of 70.

Fill in your name and set.

Example:

```
Student Name: Nellie Jackman
Student Set: 1A
[ ]  The signout buttons end the session.     (incomplete)
[x]  The .env file is NOT in your git repo.   (complete)
40/70
```

Include your self-graded checklist within your git repo in the root folder.

Deductions

The following deductions will apply if submission requirements are not met:

| | Deduction | |
|---|---|---|
| **Requirement** | **Out of 50** | **Percent** |
| Missing Hosting Link | -10 | 20% |
| Missing or incomplete Checklist | -10 | 20% |
| Missing GitHub Link | -15 | 30% |
| Video is 10s to 60s too long | -5 | 10% |
| Video is >60s too long | -10 | 20% |
| Missing Video Link | -25 | 50% |

Video Demo Requirements:

| **Your video needs to include:** | **Example:** |
|---|---|
| Your name | Patrick Guichon |
| Your set | 1A |
| Which assignment you are demoing | Assignment 3 |

| **Notes about your video:** |
|---|
| If we don't see all the functionality and features of your website demonstrated in the video, you will lose marks for not having completed this part of the website. |
| You do NOT need to show your face while recording the video, however, you must be narrating the video. Tell us what you are doing, as you are demoing your site. |
| Make sure we can hear your voice clearly, and at an adequate volume. |

| |
|---|
| Max video length: 2m:30s.<br>Your video must **NOT be longer than 2 ½ minutes**. |

| **Demo the following functionality in this order (order might vary slightly due to randomness):** |
|---|
| 1. Show the home page (/) running on Netlify (or other static hosting service). |
| 2. Click on a card and show that it properly flips. |
| 3. Click on the same card again to show that it does **not** flip again. A card should **not** match itself. |
| 4. Click on a previously matched pair card and show that it does **not** flip back. |
| 5. Show the behaviour of clicking 2 cards that don't match and show that they properly flip to the back side of the card after a short delay. |
| 6. Click on 3 cards really quickly (during the flipping animation) to show that only 2 cards can be flipped at once as expected. |
| 7. Show that the during the game, a status section is displayed with the current time, time remaining, number of clicks, number of matches made. |
| 8. Show that the game finishes with a successful "winning" message when all cards are matched. Cards should no longer be able to be flipped at the end of the game. |
| 9. Show that the game finishes with a "game over" message if the timer runs out with at least some cards left unmatched.<br>Cards should no longer be able to be flipped at the end of the game. |
| 10. Show that you can play using different difficulty levels (easy, medium, hard).<br>Show and explain what is different between each of the difficulty levels. |
| 11. Click the reset button to show that it properly resets the stats in the status section and allows for restarting the game. |
| 12. Switch from light mode to dark mode (or other themes) and back again. |
| 13. Trigger the power up. Your trigger for when the power up happens may vary, but make sure your demo video includes an explanation of when it happens and what you should see when the power up happens (example: "The power up is a bonus 3 seconds added to the timer if you successfully match 2 sets of cards in a row"). |
| 14. Briefly show the HTML file. |
| 15. Show the JS file. Explain the overall functionality of the code. Explain how the Pokémon API was used to retrieve the images for the cards. Show and explain how the status section is calculated and displayed. Show and explain the click event logic for buttons and card flipping. |