THE CAPYBARA GAME

CSE104 Project Report

A Comprehensive Application of Web Programming Skills to Architect a Simple yet Elegant Browser Game

by

BX26 Eugenio Animali
BX26 Yuki Kin

Supervisor: Damien Rohmer

CSE104 Web Programming
23 May 2024

§1: Introduction

Our Motivation

During the span of the course, we found great interest, especially in the user interaction aspect of JavaScript rather than the user interface organization aspect of HTML or CSS. For instance, the notion of class/ID, alignment and box models were extremely useful in understanding the mechanics behind what we see on our screens and helped us grasp the process of web programming, however, more dynamic coding such as canvas and animation loop in javascript provided more freedom in customization and allowed access to interaction with the user. Therefore, before the course, we held hopes to make a simple yet elegant browser game that is within the scope of the course but allow us to explore new horizons in customizing the game and going beyond the course content.

The Inspiration

Although we were set in stone in making a game, we struggled to find inspiration until we encountered the famous ‘chrome dino game’ that appears when we lose internet connection. It was a very simple game that was familiar to everyone and we immediately saw it as the primary inspiration for our project idea. In many ways, the inspiration proved to be the right one for us:

- Within the scope of contents learnt in CSE104
- Possibility to customize the aesthetics to the unique bachelor setting
- Ability to personalize the game by adding extensions
- Treatment of complicated animation coding that proved challenging

Rationale & Aim

We started our project by brainstorming rough ideas as to what content we wanted to achieve and the structure of the website. After the initial meeting, we have set ourselves the following key ideas and aspects of the website that will judge the success of our project:

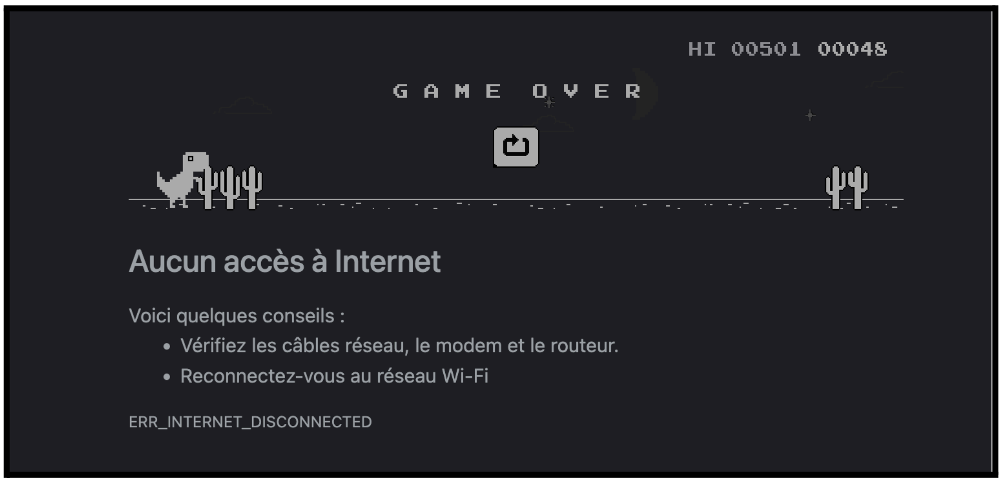
- **Perfectly functioning game**: jumping/running/ducking animation, freeze frame with game over the button, display the high score, speed-up overtime, generate obstacles at random etc
- Mimic original aesthetics and structure with a **black-and-white pixelated style**
- **Tailor to the bachelor theme**: capybaras, textbooks as obstacles etc
- Add **customization of capybaras** with different skins
- **Transition to dark mode** after a certain time
- Implementation of the **local leaderboard** with the use of cookies (beyond course)
- Option to play **background music**

Throughout the project, this will serve as a guiding criterion for us to ensure we achieve the desired results. With this website, the primary aim is to achieve a perfectly functioning game without bugs and refined details; the general aim is to produce an elegant bachelor capybara game that will be played for many years to come. As beginner web programmers, our goal is to enhance our understanding of HTML, CSS, and JAVASCRIPT, but at the same time provide a source of entertainment for the users.

§2: Background

Case Study: Chrome Dino Game

The Chrome Dino game is a browser game made by the Google UX team in 2014, it appears when the user has no internet connection. The popular game is known for many of its iconic features but is especially known for its simplicity/easy-to-understand rules and offline functionality. In this project, we will try to maintain these key advantages of the game whilst at the same time, improving and advancing the game through both adding extensions & personalizing it to a unique experience. Most importantly in the planning phase, we played the game more than enough times to conduct analysis from a web programming perspective.



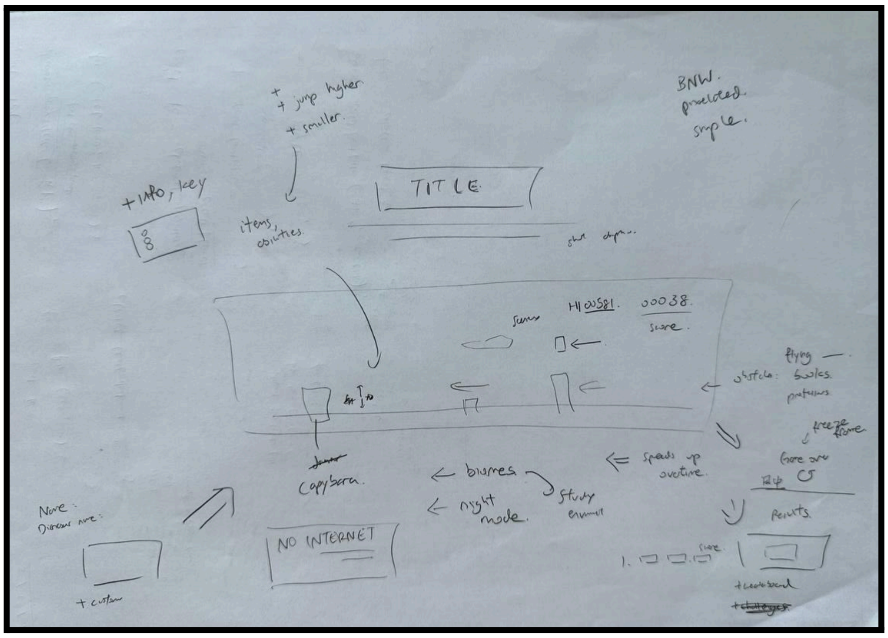
Aucun accès à Internet

Voici quelques conseils :

- Vérifiez les câbles réseau, le modem et le routeur.
- Reconnectez-vous au réseau Wi-Fi

ERR_INTERNET_DISCONNECTED

During the process, we noticed several key details that seem simple but may be challenging to code. For instance, the random placement of the obstacles with their corresponding probabilities, adjusting the detection zone of the capybara, changing all texts to white in dark mode etc. Before proceeding, we conducted a rough planning sheet to jot down our ideas on the structure of the website.



§3: Approach & Methodology

Approach

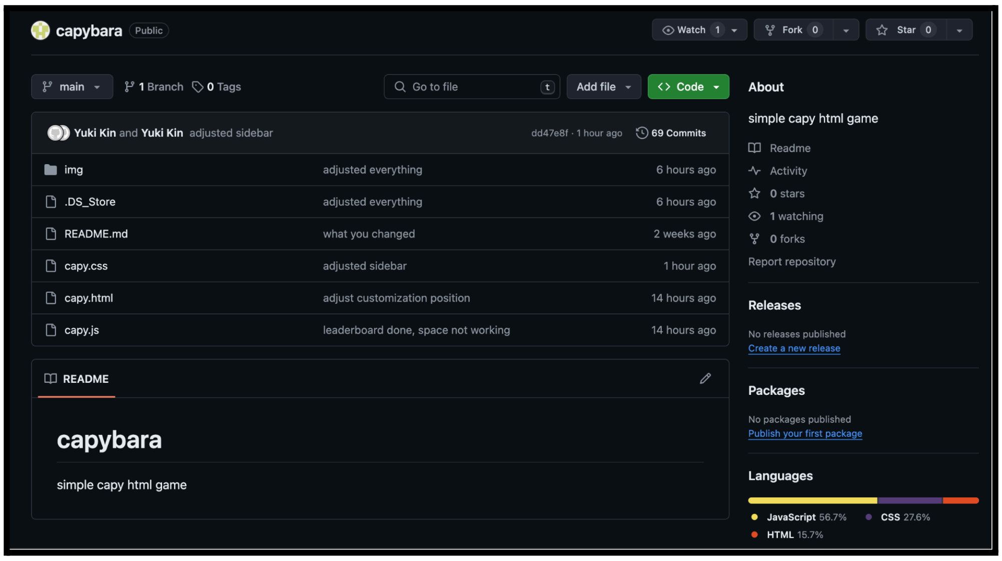
With a short span of time period to complete the project, we quickly devised our strategy at the start to lay a solid foundation on which we can work efficiently and accurately. As we are a two-person team, it is important to maintain a high level of transparency and communication between each other; whilst also identifying personal strengths of one another to divide the workload. Instead of working together throughout the projects, we decided to take the core strategy of "divide and conquer", where we work separately on separate parts of the project independently whilst combining the code frequently for updates.

Firstly, on the technical side, we quickly agreed on the basic structure and general content of the codes to ensure good practice amongst each other. This includes the use of no internal CSS, using consistent variable names, using semantic meanings, keeping HTML to a minimum for a higher reliance on Javascript, an effort to be concise etc. Next, on the logistics side, we understood scheduling may be difficult, so we set at least weekly meeting times and deadlines for each other on a document to keep track of our progress, including the use of telegram as our main communication platform.

Methodology

Initially, our goal was simple: start from the basics; we can only have a successful browser game with a solid foundation to build on top of it. We committed to making the basic functionalities work perfectly before tackling other aspects such as dark mode, customizations or music.

As mentioned before, we identified each other's strengths and interests; dividing the parts accordingly. For instance, with Yuki's background in social media & graphic design, Yuki worked mainly on HTML first to ensure the structure of the website is in the right format and style. Eugenio, on the other hand, preferred working with the game's functionality itself, therefore started by experimenting with events and animation loops early on. Throughout the method, we decomposed our progress into smaller achievable goals to ensure we were on track and on time.



Yuki Kin and Yuki Kin adjusted sidebar dd47e8f · 1 hour ago 69 Commits

img	adjusted everything	6 hours ago
.DS_Store	adjusted everything	6 hours ago
README.md	what you changed	2 weeks ago
capy.css	adjusted sidebar	1 hour ago
capy.html	adjust customization position	14 hours ago
capy.js	leaderboard done, space not working	14 hours ago

README

capybara

simple capy html game

About

simple capy html game

Readme
Activity
0 stars
1 watching
0 forks
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● JavaScript 56.7% ● CSS 27.6% ● HTML 15.7%

However, most importantly, we quickly realized the difficulties of working remotely/separately on the same files, since there may be a lack of consistency between the two files and merging may cause the loss of minor changes on each side. Therefore, we **implemented a repository on Git Hub** where all the files including code, images etc will be stored. By doing so, we can swiftly merge code files and update remotely without the need to locally transfer files to each other. Although the process needed some time to get used to ("git pull", "git commit", "git push" etc), the method proved very useful and efficient in the long term. Additionally, Git Hub provided analysis on the project such as the ratio of the codes with javascript being 56.7% of the code (see figure above).

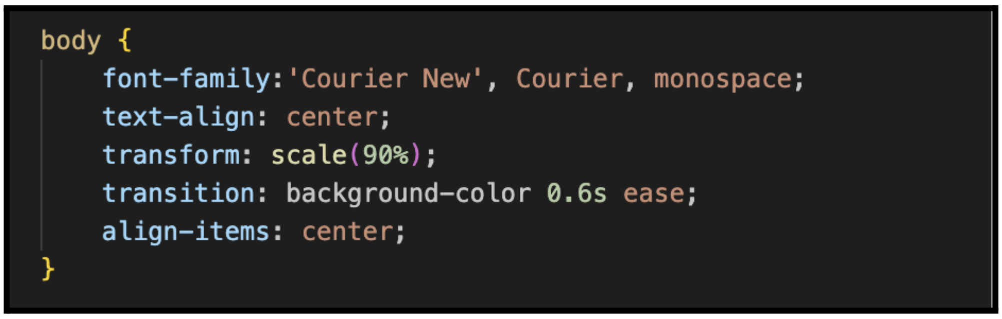
However, sometimes when we happen to be working at the same time remotely, the layers of "git pull" and "git push" may create various "branches" that can be confusing to merge. Therefore, we took extra attention to labelling each branch with what we did using **"git commit -m"** as well as used a **live-share add-on in Visual Studio Code** that allowed for editing at the same time.

§4: Records & Analysis

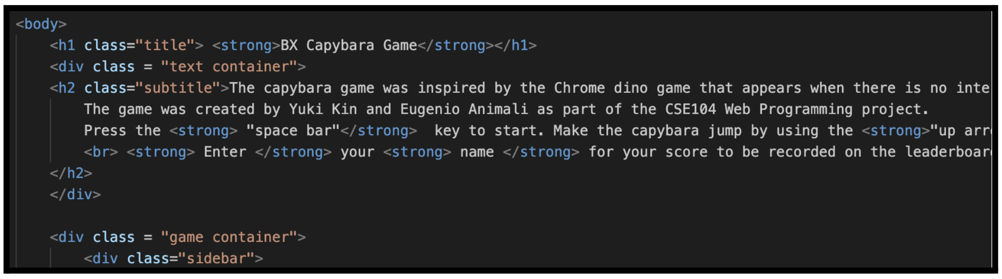
We encountered many interesting hurdles in our pursuit of a more complete, interactive running game that did not discourage us, but rather taught us important lessons that we could use to write better code.

Retro Aesthetics

From brainstorming the game, we understood the importance of the overall aesthetic of the game and its stylistic structure. We wanted to achieve the 'retro pixelated' style of the original chrome dino game that features a black and white colour scheme, pixelated objects and a standard typewriter font. Maintaining a consistent style is crucial to enhance the game's quality and experience. To start, I laid out all the elements needed for the basic game in the HTML file with divs labelled with clear classes to avoid confusion. I quickly realized the effectiveness of this approach as it allowed me to adjust the structural elements separately. But most importantly, to ensure consistency of the style in the game, instead of adding a property to each CSS selector, I added properties "font-family", "text-align" and "transform: scale" to the "body" selector, which applies the desired style to every element efficiently.



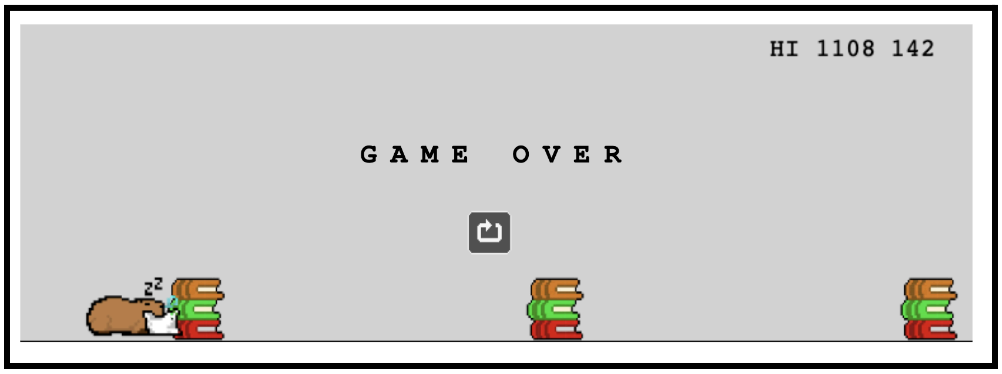
```css
body {
    font-family:'Courier New', Courier, monospace;
    text-align: center;
    transform: scale(90%);
    transition: background-color 0.6s ease;
    align-items: center;
}
```



```html
<body>
    <h1 class="title"> <strong>BX Capybara Game</strong></h1>
    <div class = "text container">
    <h2 class="subtitle">The capybara game was inspired by the Chrome dino game that appears when there is no inte
        The game was created by Yuki Kin and Eugenio Animali as part of the CSE104 Web Programming project.
        Press the <strong> "space bar"</strong>  key to start. Make the capybara jump by using the <strong>"up arr
        <br> <strong> Enter </strong> your <strong> name </strong> for your score to be recorded on the leaderboar
    </h2>
    </div>

    <div class = "game container">
        <div class="sidebar">
```

The Bachelor Theme

In many ways, we tried to integrate the bachelor theme into the game to make it more personal for the users. Our initial ideas were all successfully achieved in the final version of the game, including the implementation of the capybara figure (bachelor mascot character) with obstacles of thick textbooks and pens that align with the theme since bachelor students pursue a rigorous double major degree. The completed game can be seen in the figure below:



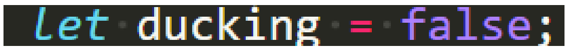
Whilst in CSS/HTML, the basic structure & content were established, we heavily utilized JavaScript to architect a solid foundation of the game that we continued to refine throughout the process. More precisely, we mainly implemented the following functions: (brief overview)

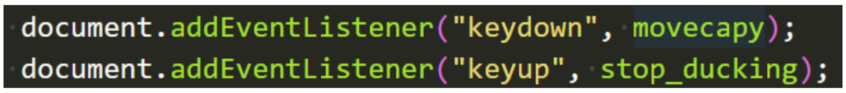
- **startgame()** {initializes the game settings and starts the game loop}
  - Canvas Setup: by setting its dimensions and drawing a capybara image
  - Game Loop: call update() function using requestAnimationFrame to start the loop
  - Obstacle Placement: set interval to call placebook() every second
  - Event Listener: add events listeners to "key-up" and "key-down"
- **update()** {handles the main game loop, checking for game over conditions}
  - Running Toggle: Toggles the running state every 7 frames for animation purposes
  - Duck Handling: Updates the capybara's image and position based on whether it is ducking or running
  - Canvas Refresh: Clears and Draws the Capybara at its current position
  - Move Obstacles: Updates the position of obstacles and draws them on the canvas.
  - Collision Detection: Calls detectCollision to check for collisions between the capybara and obstacles
  - Game Over Handling: Manages the end game conditions, updating the leaderboard, showing game over text, and stopping the game loop
- **placebook()** {generates obstacles at random intervals and adds to array}
  - Game Over Check: Returns early if the game is over
  - Obstacle Creation: Creates a new obstacle with random properties based on a random chance
  - Array Management: Removes the oldest obstacle from the array if there are more than 5 obstacles to prevent memory overflow
- **detectCollision()** {checks for collisions between the capybara and obstacles}
  - Bounding Box Calculation: Defines precise bounding boxes for each obstacle
  - Collision Detection: Iterates over the obstacles and checks if the capybara's bounding box intersects with any obstacle's bounding box
  - Game Over: Sets the gameOver flag to true if a collision is detected and updates the high score if needed

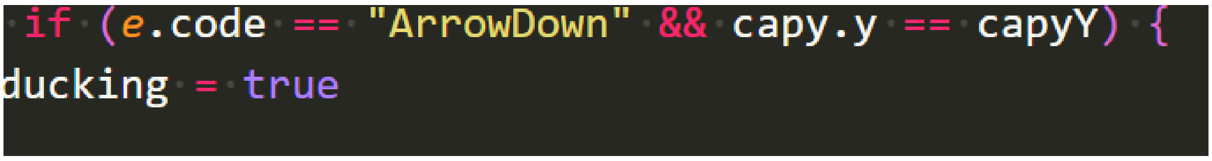
Ducking

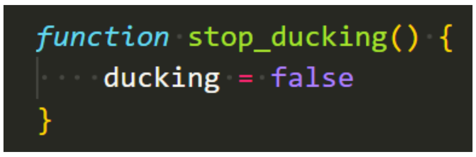
How to implement the ducking capybara was not obvious at first. The first thought was to check whether a key was pressed during every frame, and from there calculate whether the capybara was to duck or not. However, after a first implementation of this, we realised that due to the asynchronous running of functions and event listeners in javascript, the frames would not be perfectly in sync with the arrows, and the capybara would jitter as the key was being pressed.

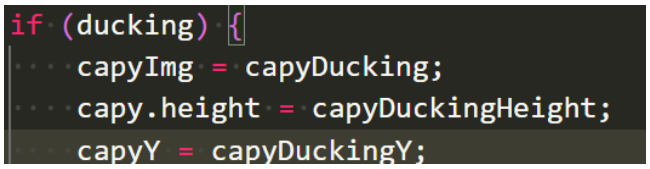
We then implemented a different approach that does not make the calculation happen every frame, meaning that the capybara would not jitter when ducking, and the animation was smooth. The new implementation had one event listener for “key-down” and “key-up” respectively, which would change a global boolean variable. this would be considered by the update() function to know what capybara to show. Here is some of the code that makes this work.


```
 let ducking = false;
```


```
document.addEventListener("keydown", movecapy);
document.addEventListener("keyup", stop_ducking);
```


```
 if (e.code == "ArrowDown" && capy.y == capyY) {
ducking = true
```

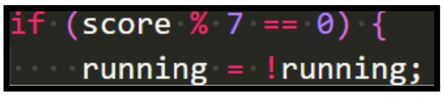

```
function stop_ducking() {
    ducking = false
}
```

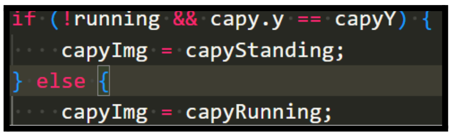

```
if (ducking) {
    capyImg = capyDucking;
    capy.height = capyDuckingHeight;
    capyY = capyDuckingY;
```

You may notice in the last image we had to implement a variable for the normal y position of the capy, as it had to go lower when ducking. Before this, capy.height and capyY were fixed values that did not need to be manipulated.

The Running Problem

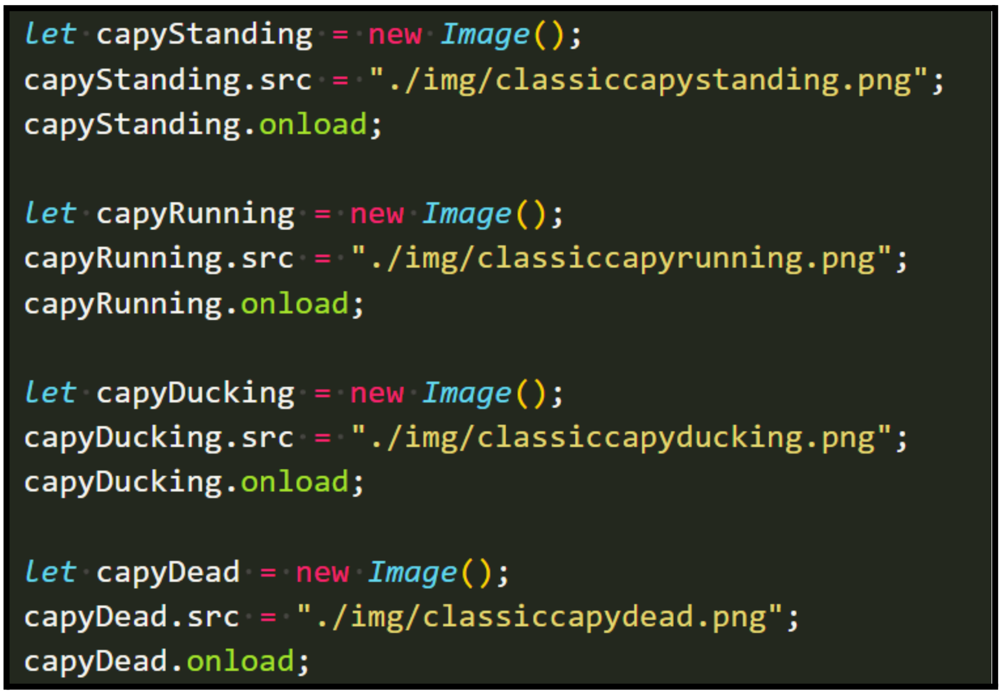
In our project, we implemented an animation mechanism for the capybara character that alternates between two running images, creating a fluid running motion. This was achieved by switching the images every n frames using the following code snippet: This condition toggles the running state every 7 frames, allowing us to alternate the displayed running images. Additionally, we incorporated logic to ensure the capybara only shows a jumping image while airborne: This code checks the running state and the capybara's vertical position, capy.y, against its ground position, capyY. If the capybara is not running and is on the ground, it displays the standing image; otherwise, it displays the running image. This implementation ensures a smooth and realistic animation cycle for both running and jumping actions.


```
if (score % 7 == 0) {
    running = !running;
```

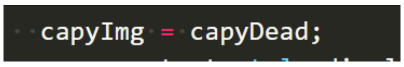

```
if (!running && capy.y == capyY) {
    capyImg = capyStanding;
} else {
    capyImg = capyRunning;
```

Onload

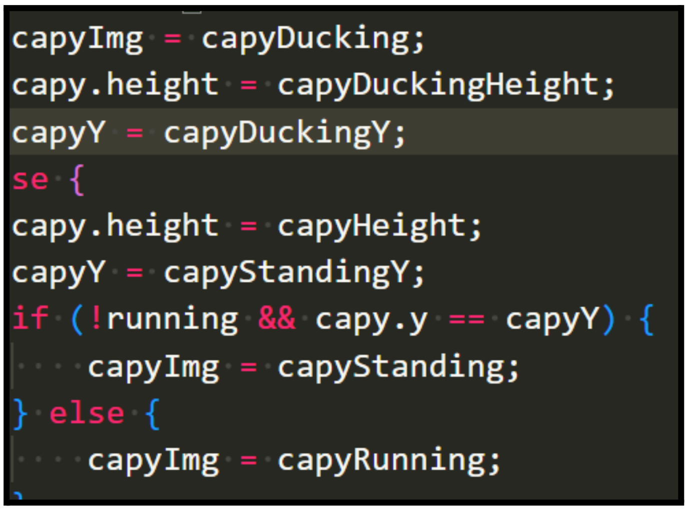
We had some trouble understanding the ".onload" method, as we learned JavaScript does not run line by line. When an image source is chosen, the image must be loaded before being drawn on the canvas. This caused problems at the beginning as we had to onload new images when we wanted to change the image to the dead capybara image (this was the only image change we were implementing at the time). However, we did not know how to onload conditionally only when we had to change the image. What was being stored was only the path strings to the images, meaning they would have to be loaded continually. We solved this problem by loading all the image sources at the beginning and storing them directly in variables so that they can be accessed each time without loading. We then assign the loaded image objects to the capyImg variable as you can see below.



```javascript
let capyStanding = new Image();
capyStanding.src = "./img/classiccapystanding.png";
capyStanding.onload;

let capyRunning = new Image();
capyRunning.src = "./img/classiccapyrunning.png";
capyRunning.onload;

let capyDucking = new Image();
capyDucking.src = "./img/classiccapyducking.png";
capyDucking.onload;

let capyDead = new Image();
capyDead.src = "./img/classiccapydead.png";
capyDead.onload;
```



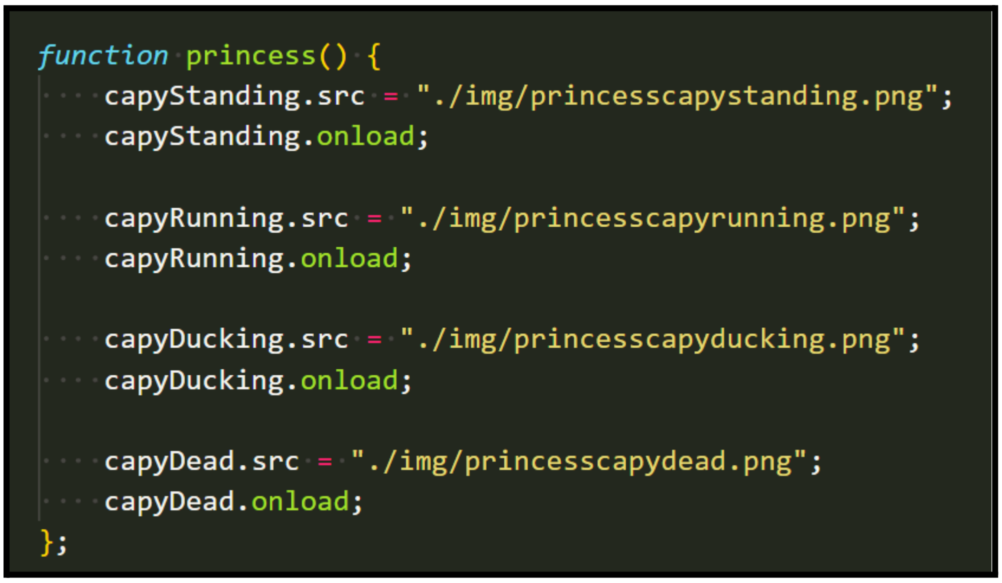
```javascript
  capyImg = capyDead;
```



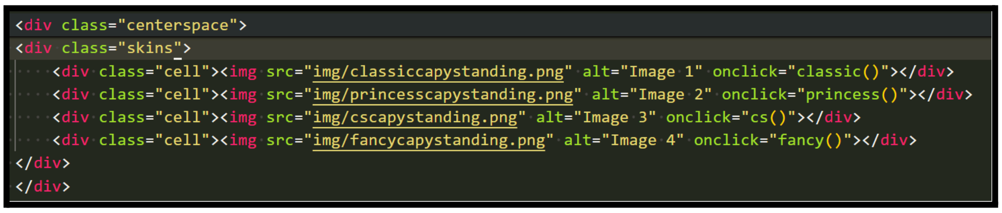
```javascript
capyImg = capyDucking;
capy.height = capyDuckingHeight;
capyY = capyDuckingY;
se {
capy.height = capyHeight;
capyY = capyStandingY;
if (!running && capy.y == capyY) {
    capyImg = capyStanding;
} else {
    capyImg = capyRunning;
```

Customisation

In our HTML/JavaScript/CSS project, we successfully implemented a feature allowing users to select different skins for capybaras in our game. This was achieved by utilizing HTML <div> elements, which were assigned specific event listeners. These listeners triggered functions that modified global variables corresponding to the various skin images. The selected images were then dynamically rendered within the game's animation loop using the requestAnimationFrame method. By manipulating these global variables, the game could seamlessly update the Capybara skins in real-time, providing a smooth and customizable user experience. This approach effectively separated the skin selection logic from the core animation process, enhancing both the modularity and maintainability of the codebase. The images we draw one by one in pixel art, giving room for some artistic creativity which is essential for a game like this, and lets us play with dynamically assigning images. However, we had to make sure all images were loaded and cropped correctly at the start of the game. We were able to tailor the skins to the bachelor experience, featuring the "CS major" skin that resembles some of the people you may have encountered walking into your class. Pyjamas and a dirty hoodie! There are also the "straight-A students" that are able to dress fancy to class no matter the occasion, and the princesses; those you almost never see working, only having fun. The goal became that of avoiding textbooks, and if you get hit by one, you fall asleep.



```javascript
function princess() {
    capyStanding.src = "./img/princesscapystanding.png";
    capyStanding.onload;

    capyRunning.src = "./img/princesscapyrunning.png";
    capyRunning.onload;

    capyDucking.src = "./img/princesscapyducking.png";
    capyDucking.onload;

    capyDead.src = "./img/princesscapydead.png";
    capyDead.onload;
};
```

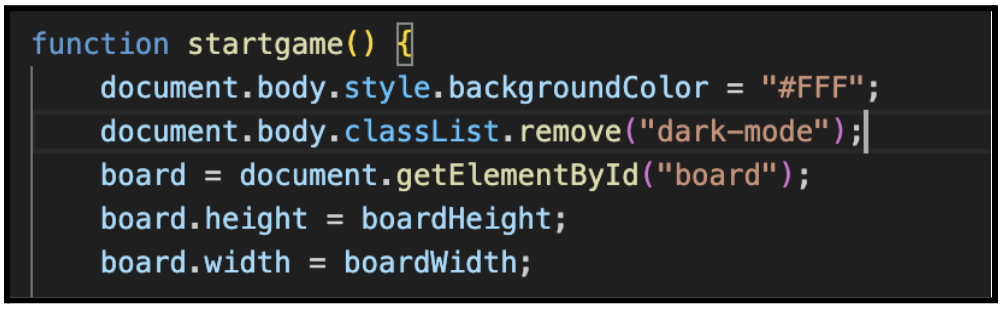


```html
<div class="centerspace">
<div class="skins">
    <div class="cell"><img src="img/classiccapystanding.png" alt="Image 1" onclick="classic()"></div>
    <div class="cell"><img src="img/princesscapystanding.png" alt="Image 2" onclick="princess()"></div>
    <div class="cell"><img src="img/cscapystanding.png" alt="Image 3" onclick="cs()"></div>
    <div class="cell"><img src="img/fancycapystanding.png" alt="Image 4" onclick="fancy()"></div>
</div>
</div>
```

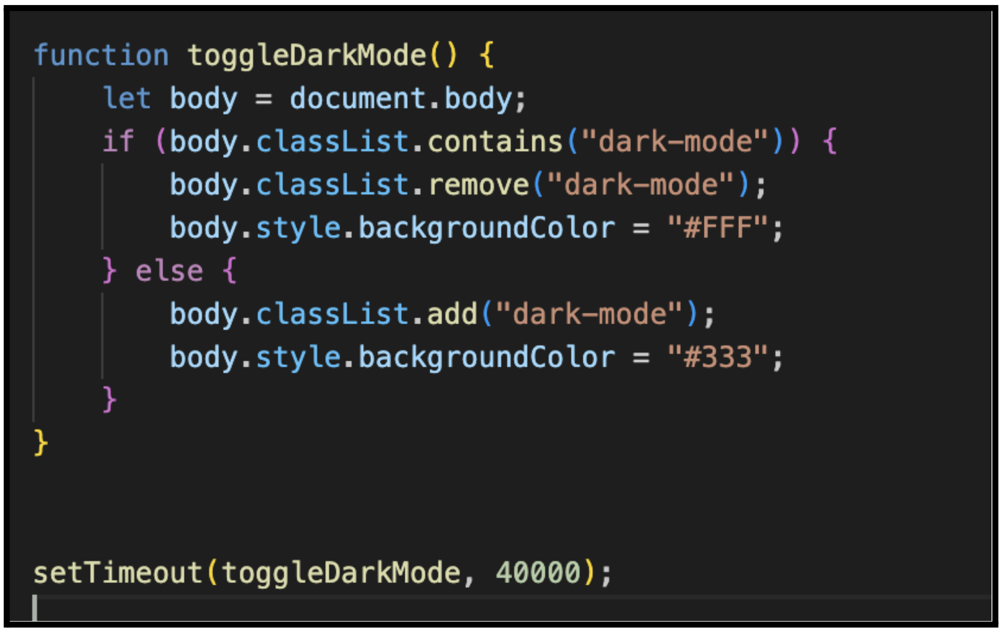
Dark-Mode

Like the original Chrome Dino game, one of our main goals in this game was to implement the 'dark mode' where the game goes into 'dark mode' after a certain amount of time. The dark mode features a black background with every element in the website inverted to the white colour to accommodate it.

Initially, we struggled to find a consistent way to implement the dark mode since the task is not as simple as inverting the colours of the selector "body" but needs to be implemented individually to each element to ensure it is done in the right fashion. To start with, we experimented with making a single function for dark mode that includes all details on the CSS style of the elements in javascript using setTimeout to activate it after a certain amount of time, however, this proved useful until we saw frequent delays and discrepancies in particular elements. Instead, we improved the code by making a class dark mode in CSS and adding/removing it to the document.body in JavaScript. This allowed us to both simplify the javascript functions and manipulate the dark mode style for each element separately in CSS. Additionally, we added transition time into dark mode so it's more natural and less artificial by adding "transition: background-color 0.6s ease;" to the body selector in CSS.



```
function startgame() {
    document.body.style.backgroundColor = "#FFF";
    document.body.classList.remove("dark-mode");
    board = document.getElementById("board");
    board.height = boardHeight;
    board.width = boardWidth;
```



```
function toggleDarkMode() {
    let body = document.body;
    if (body.classList.contains("dark-mode")) {
        body.classList.remove("dark-mode");
        body.style.backgroundColor = "#FFF";
    } else {
        body.classList.add("dark-mode");
        body.style.backgroundColor = "#333";
    }
}


setTimeout(toggleDarkMode, 40000);
```

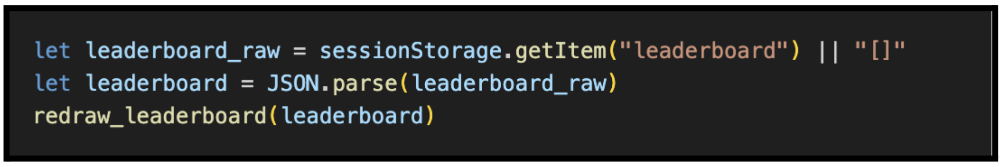
This aspect was easy to start but proved surprisingly challenging to perfect in the long term as we added more elements. We frequently encountered problems such as text colour not turning white at the same time, dark mode not cancelling out when restarting the game etc. Therefore, we are proud of what we did for dark mode as its efforts are minor but cumulative, as seen after 40 seconds of playing.

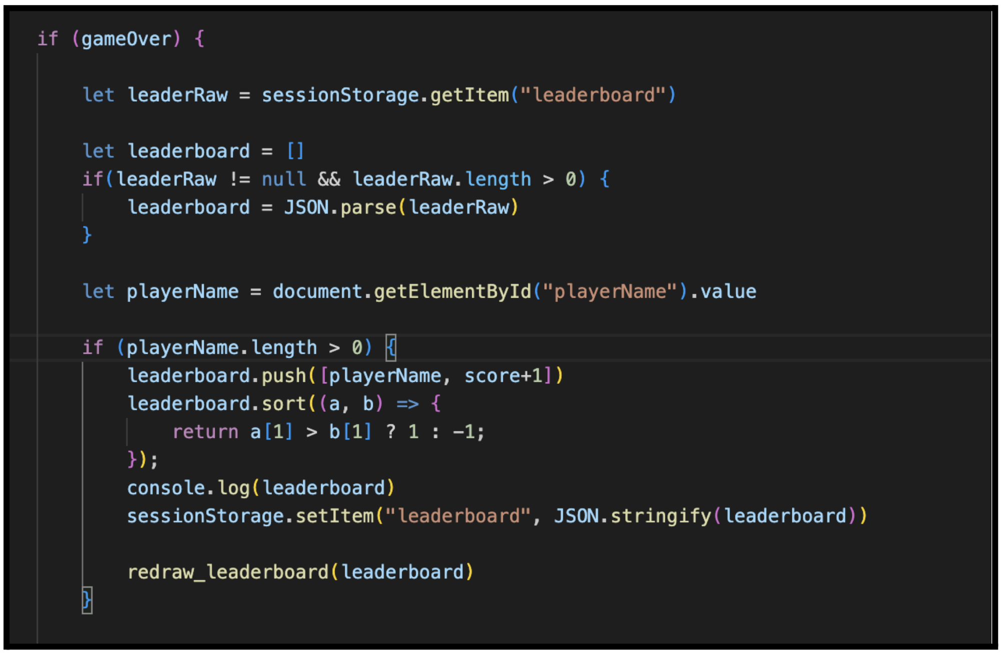
Leaderboard

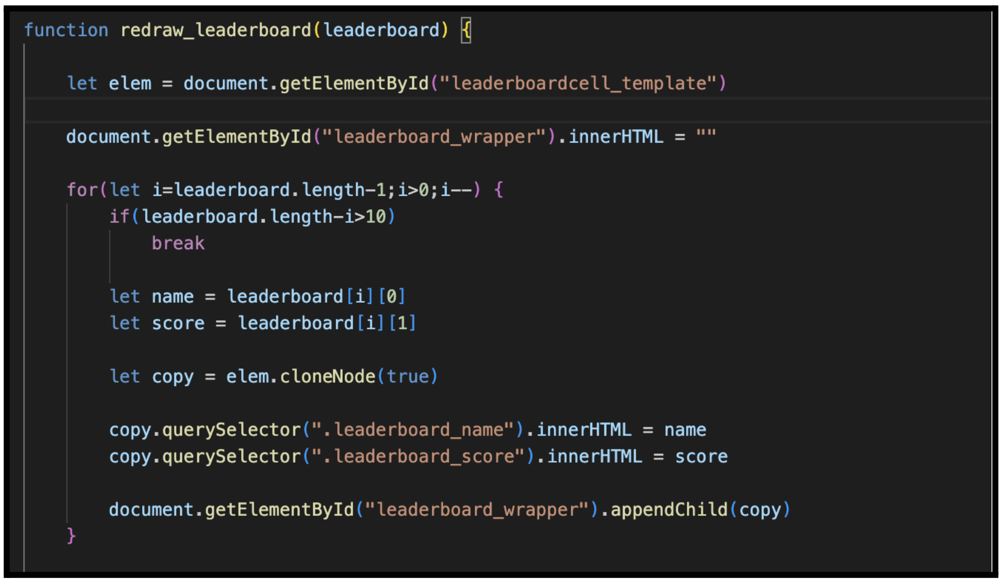
A potential idea we had during the process of this project was to add a leaderboard to the website that displays the top 10 scores. This is not part of the original Chrome Dino game and will provide us with a challenge since it goes beyond the CSE104 course content with the use of cookies and libraries to record scores even after refreshes. Although discouraged from doing so, we proceeded to experiment with a new tool: JSON and the use of cookies.

Since the implementation of a ‘ global all-time ‘ leaderboard will require external servers and we do not have the resources, we decided to focus on a local leaderboard that will be preserved in the form of the user’s local cookies (disappear if cookies are cleared). We conducted research on such a method and encountered the use of JSON (Javascript Object Notation) which is used for the representation of data structures and objects.

We are especially proud of this aspect of the game since it was beyond the CSE104 course content and it resulted in functioning successfully, with a local leaderboard for each local device that can track the user’s highest scores.


```
let leaderboard_raw = sessionStorage.getItem("leaderboard") || "[]"
let leaderboard = JSON.parse(leaderboard_raw)
redraw_leaderboard(leaderboard)
```


```
if (gameOver) {

    let leaderRaw = sessionStorage.getItem("leaderboard")

    let leaderboard = []
    if(leaderRaw != null && leaderRaw.length > 0) {
        leaderboard = JSON.parse(leaderRaw)
    }

    let playerName = document.getElementById("playerName").value

    if (playerName.length > 0) {
        leaderboard.push([playerName, score+1])
        leaderboard.sort((a, b) => {
            return a[1] > b[1] ? 1 : -1;
        });
        console.log(leaderboard)
        sessionStorage.setItem("leaderboard", JSON.stringify(leaderboard))

        redraw_leaderboard(leaderboard)
    }
```



```
function redraw_leaderboard(leaderboard) {

    let elem = document.getElementById("leaderboardcell_template")

    document.getElementById("leaderboard_wrapper").innerHTML = ""

    for(let i=leaderboard.length-1;i>0;i--) {
        if(leaderboard.length-i>10)
            break

        let name = leaderboard[i][0]
        let score = leaderboard[i][1]

        let copy = elem.cloneNode(true)

        copy.querySelector(".leaderboard_name").innerHTML = name
        copy.querySelector(".leaderboard_score").innerHTML = score

        document.getElementById("leaderboard_wrapper").appendChild(copy)
    }
```

§5: Conclusion

Throughout this project, we have successfully created a simple yet elegant browser game inspired by the Chrome Dino Game, tailored to reflect the unique theme of bachelor life. We are proud of the dynamic gameplay features, such as the jumping, running, and ducking animations, as well as the incorporation of character customization and dark mode transitions. By leveraging HTML, CSS, and JavaScript, we not only enhanced our technical skills but also learned valuable lessons in collaborative coding, problem-solving, and using tools like JSON for data management and GitHub for version control. Our final product, complete with a local leaderboard and background music, stands as a testament to our dedication and ability to go beyond the course content, providing an engaging and polished user experience.

THE END