CSE 104
Rémi Guillou

Final project website Report

Preface:

This document is my report for my website that I created as the final project in the course CSE104: Web Programming (2022-2023).
Here I will go over the steps I took in order to finish this project, my thought process and an overview of the website and its functionalities.
Here is a link to the github repository for this project:
https://github.com/Remix375/simulation
(Note that I may still work on the project after handing it in so the files may be different then those handed in)
Here is a link to the website uploaded using Github Pages:
https://remix375.github.io/simulation/

Introduction:

My initial idea when presented with this project was to create a chess website. It would have been pretty basic and I was aiming to use an existing API for the chess logic. lichess.com has a great API that I could have used for this. You could play against the bot at different levels.
However most people were making games and it would have been a bit boring and classical. Therefore I decided to make something else.
The js exam gave me the idea to make a physics simulator as I believed it would be fun. I have often interacted with physics simulators in the past, for instance in game engines. However I have never made one from scratch and I thought that it would be a good experience that would enable me to better understand how they work.
This attempt is very basic, using only spheres and lines as shapes. I also heavily relied on online documentation for collision detection and reaction (all documents used are linked to in the appendix at the end of the document).

Part I: Initial Idea

My first plan for the website was pretty simple: I would create a physics simulator. Over the course of a week I refined it by reading articles on the subject and determining what was a realistic goal for the time I had. I had to figure out what the user could simulate and how he would interact with the website.

I ended up with 2 types of interactions: objects that could be placed in the scene and global scene settings.
For the objects I decided to go with spheres, platforms, magnets and sphere generators.
The spheres would have a mass, color and size, magnets a strength, mass and size, generators a rate and platforms simply a position as they wouldn't move.
For the scene settings I had gravity and air friction. (I would then add collision elasticity)
I also wanted to be able to pause the scene and select elements in it to modify them.

At the beginning I also wanted to make a game using the physics simulator. There would be a starting block and an end block. You could change some settings in the scene and when starting the game, the start block would output a ball. The objective would be to tweak the setting so that the ball touches the end.
I also wanted to make a "level creation" page where the user could create a level of the game and he would get a string that could be pasted on the website by anyone to play the level.
However I didn't have time to implement all of that. But I think that I will continue working on it and finish it one day.

Part II: Basic Layout

To start off this project I needed to decide on a layout. I decided to put the settings on the left hand side and the scene on the right. I also kept some place under the scene for useful buttons.
I then chose colors for the background and a font for the text.
The settings would be divided in Two: the objects and settings for the scene. The user could switch between the two by clicking on two buttons.
I then tackled the hardest part of the website: displaying this vertical line correctly:

Part III: Creating the canvas

For displaying the simulation I would be using a canvas element. I set it up and wrote a bit of javascript for it in the file “logic.js”. I created an animation loop “`mainLoop`“ that would erase everything on the canvas and redisplay the next frame. I will also need to update every element on screen in this loop.
First thing I did was implement a “zooming” feature on the website. When you scroll, the scene will zoom in or out. The zoom is only one dimensional, it zooms in on the top left. I could probably make it zoom at different places depending on the position of the mouse in the future. The way it works is that scrolling changes a global “`current_zoom`“ variable that is passed in to every element when they are displayed. Their size is computed depending on it.
I also display a scale for 1 meter on the bottom left. The function for that is in the file “objects.js”. All the classes for objects on screen will be in that file.

Part IV: Objects

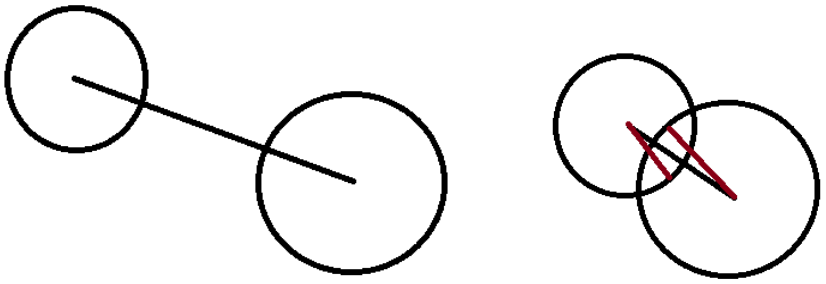
In the file “objects.js” are the classes to create instances of every different object:

- Vector: useful class for position, velocity and acceleration
- Circle: Parent class for circular object
    - Ball: child of circle, create a Ball.
    - Magnet: child of circle, create a Magnet that attracts balls that are around by accelerating them towards it
- Wall: create a wall
- Generator: creates ball generator

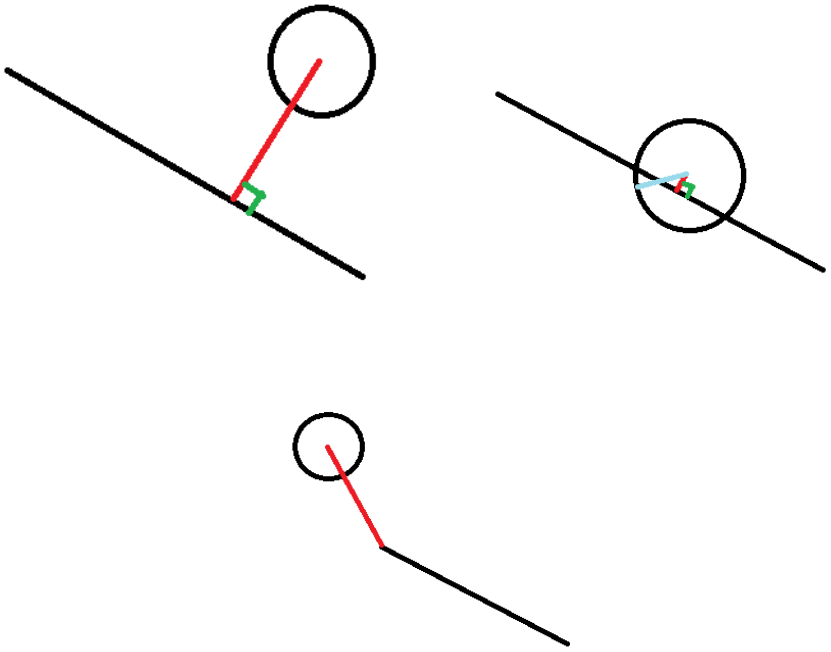
All classes except Vector have draw, update and die methods.
We also have arrays (`circles balls magnets generators walls`) in logic.js in which we store instances of the classes that are onscreen. We iterate over them in the mainloop, display and update them.

Part IV: collision detection and reaction

First detection: for spheres, we just need to get the distance between them and check that it is greater than the sum of their radiuses.



For a sphere and a line, we get the closest point on the line to the sphere center and check that it is greater than the radius of the sphere.



For reaction, the first step is the penetration resolution. As the action isn't continuous, there are a finite amount of frames, the two objects colliding will be inside each other so we first get them out and then change their velocity depending on their velocity before, their masses and the elasticity.
If the elasticity is 1, the collision is completely elastic, no energy is lost. If it is 0, the collision is inelastic so the objects will stick.

Part V: Display and storing data

To create an object you click on one of them, input their settings and press in the scene. One challenge I faced while implementing that was storing data, displaying it and hiding it.

I actually used 2 different ways of doing that. All in the file param_dropdown.js For the objects, I created 4 functions (`create_ball_html create_magnet_html create_generator_html create_wall_html`) that generate and return the html to create the fields with the data in them. I did it that way because I would use those functions later too. The data is stored in the variable `input_data` as an object. I then modify that data as the user inputs new values.

The second way of doing things is much easier in my opinion, I used it to switch between Objects and Scene settings. The html elements are already in the DOM but their css is set to "`display: none`". We simply switch which one isn't displayed on click.

To select elements in the scene and modify their properties, I reused the functions (`create_ball_html create_magnet_html create_generator_html create_wall_html`) and changed different data with them.

Part VI: Small technical details

The force of the magnet is inversely proportional to its distance to an object.
The fps counter on the top left calculates the number of updates to the scene per second. It is calculated using the time between 2 consecutive frames.
If an object is initialized with a mass of 0, it will be immovable. This can be useful especially with magnets.
Pause button on the bottom, pauses the scene. Reset button just next to it, clears the whole scene.
Size of objects is the radius in centimeters. Mass is in Kg.
Gravity is in m/s^2.
Objects outside the scene are deleted every frame.

Conclusion:

This project is far from perfect. I could have done a lot more but ran out of time. It was still a lot of fun to make and I will probably continue working on it in the future. The code is messy and I could have done some optimizations but it gave me a first look into what goes into making a physics simulator and I am happy with what I ended up with.

Appendix:

Resources used:

Main resources for debugging:

https://stackoverflow.com/
https://developer.mozilla.org/
https://www.w3schools.com/
https://css-tricks.com/

Physics simulator:

https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects
https://developer.mozilla.org/en-US/docs/Games/Techniques/2D_collision_detection
https://youtube.com/playlist?list=PLo6lBZn6hgca1T7cNZXpiq4q395ljbEI_
https://github.com/danielszabo88/mocorgo
https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection