City Simulation Final Project Report

Adyan, Bilal, Zain, Elias

April 24, 2024

1 Introduction

City Simulation is an idle game primarily using Java Spring, JavaScript, HTML/CSS, and Three.js. The game consists of a 3D model of a piece of land and a menu to the side. You can press the "Generate Profit" button to get revenue in order to buy the props in the shop. You can buy the props in the shop to add to the 3D model. The props you can buy consist of Buildings and Vehicles. Buildings generate money passively per second. You can only buy the next props after you have purchased the previous props. Vehicles add the amount of money you receive after each click. The goal is to buy everything in the shop to win.

2 Motivation

Our motivation for this project was to make use of Java's main characteristic of "Object Oriented Programming." This is something that I will go over in the Code Explanation section, but the way in which the props in the shop are given is due to the objects created in the Java server side file. Moreover, we wanted to make something that was semi-fullstack. Packaging a web application with a frontend and a backend was important so we could understand how modular we could make our projects in the future.

3 Core Code

Like I mentioned before, we used Java Spring for the backend and basic HTML/CSS and JavaScript for the frontend. For starters, Java Spring works on MVC logic for its REST API. This means we had set up two controllers. WebController controls what HTML page is shown based off user interactions with the page. GameController is the actual meat of the game-play.

Before we get into that, the POJO classes need to be explained. We have two class types: Money and Props. Money is a class that contains details about the user's money. After interacting with the game, the money in here increases, decreases, or has its multiplier and passive changed based off if they purchase a building or vehicle. Props is a class that consists of all the items that are in the shop. Props is actually a superclass with Buildings and Vehicles as its subclasses. All props share similar traits such as a name, price, and ID (we will get to this later on why we need this). However, props differ in that they either give money passively or add to the amount of clicks for the user. For this reason we decided to create a class hierarchy to make the process more modular and easier to keep up with. One more extra value we added was the "type" value in order to differentiate between a Vehicle object or a Building object in JavaScript.

JavaScript makes API fetch requests from the file called "game.js" to the GameController file. Since GameController is a Rest Controller, it initializes when the program starts. Eight props are initialized using the concept of "polymorphism." Since Vehicles and Buildings are both Props, we can create a Vehicle or Building object under Props. This is important since we want to create an ArrayList that contains both Vehicles and Buildings. In the GameController constructor, we add the created objects to the ArrayList. When the game page is loaded, game.js calls the initialize-shop endpoint that is declared in GameController that returns the ArrayList to it. We make sure to explicitly set the multiplier and passive to its initial state as well as

reset the money to zero in case the user reloads. Then a function in game.js iterates through the array to initialize the shop. It makes use of the ID in order to index each prop. This is important since we need to know what the last prop in the array is (and we are using ForEach to iterate so we cannot index normally anyways). The JavaScript file makes API fetch requests to the other endpoints to add, subtract, and change the multiplier or passive which alters the Money object that was initialized in GameController.

The performance is streamlined and only experiences hiccups due to the Three.js models implemented into the website. There is also an event loop that takes slower to update the actual number on the screen due to problems with Java Spring and JavaScript interactions. Overall, the project performs very well.

4 What we learned

We learned the power of Object-Oriented Programming in creating this project. It helped us make the program much more modular than if it were without objects/classes. The idea of polymorphism also helped when adding the objects into a single ArrayList. One clever thing that we found out was normal was to make a class that dictates the user interacting the site. Having a Money object that constantly updates through client-side interactions saved a lot of headaches of having different "money" variables floating around in our code.

5 Links

Github: https://github.com/adyan1025/city-sim/tree/master
Deployed Website: https://citysim-40bc545787a6.herokuapp.com/