Factory Manager Summary

Group 28 - Dongxin Zhang, Tao Wang, Luke Repta, Mike Apreza

Factory Manager is a business simulation web game where the player must manage and expand their empire of factories. The player starts in the first region with a startup fund and free-to-build factory. They have to manage their resources to expand their influence and earn revenue. Each region has their own set of factories, products to manufacture, resources available, and delivery trucks. All unlocked regions contribute to the player's overall revenue. Each factory has their own unique set of machines used to produce products, which consumes resources, and produces region-shared products. The delivery system allows the selling of products the player has in stock to earn money and buying of resources, which costs money, to build products. To win, the user must have the minimum of one million dollars and earn a thousand dollars per unit of time. Should the player earn a negative net-worth over a certain period of time, they lose. The final score is calculated according to the time it took to win and how much money they possessed.

The project is based on the work from Aashish Agrawal, Karan Ahuja, Richard Miramontes, and Zohar Sajith (Group 2) from Fall 2019: Group 2 - Factory Manager - Final Report.pdf.

First Release

The tutorial mode has a message box welcoming and guiding the player up to reaching the "Machines" page. The welcome page has disabled side buttons, three unbuilt factories, and some header information. The Machine page kept the header and displayed a research and back button that functioned. The player could select the "Basic Machine" and a product, which would trigger the incrementation of "Products." The Research page had nodes of different kinds of guitars that could be researched. Beside the starter node, each one had a prerequisite. A lot of items were hard-coded and logic was missing.

Second Release

A new welcome screen allowed the user to play the game with or without the tutorial. Without the tutorial, the player is able to buy all the factories given the money restriction. The side buttons, besides "Machines" which require the selection of a factory, are enabled. The Research page is available through the game page. The research of nodes took money and time, however, it was hard to tell when it was done. The updated Machine page now displayed the single product being manufactured and resources required for it. The blueprint dropdown enabled the selection of only one product to be made in the factory. Machines could no longer produce different types. The Plan Deliveries page had "Stores" and "Resources" buttons that popped up modals where the user could sell Products and buy Resources, respectively. Every dropdown in the game had options that were available to them based on product quantities and researched nodes. Everything was dynamically created/loaded. A single file dealt with time-related actions. Another handled real-time updates. A winning condition was added, too. The GameState class holds all the information a user needs in every region.

Comparison with Original Project Design Document

We designed our interface around the design proposed in the original design document, though we replaced the “Build Items” button with the research button. We also excluded the “Personnel” button, as we did not implement this feature, instead opting to create a more streamlined and user-friendly production system. Our production system simplifies the original report by having a single-shared product being produced by all machines in a factory. The Region system was also streamlined, by having every factory in a region share products/resources and having the Delivery system be used per region rather than per factory. Our implementation of the Research system is more in depth as it allows for researching new machines and bonuses, rather than just new products. The original document intended for a server with a database storing user log-in information alongside saves and high-scores, but we chose to focus on creating the game itself instead of implementing the server. However, we created the groundwork for making game saves by having a single unified “GameState” object that stores all the information about the game’s current state, for possible saving and later reloading.

Testing and Inspection

For testing and inspection, Luke submitted researchSystem.js, Mike submitted deliverySubsystem.js, and Dongxin submitted WelcomePage_SuppliesModal.js. In general, the inspection of each other’s code found that we should write more detailed and number of comments, use more consistent and easy-to-read style, and better functionalize large blocks of code. Testing was done by starting a new instance of the game, then following the exact input procedure described to test how the code performed in the tested situation.

Recommendations and Conclusions

In our project retrospective, we identified many areas of our workflow that were suboptimal and could be refined to better the pace and quality of our work. We initially started out with as few as one meeting a week, which took a while to identify as not being sufficient. We also found that we should have had more meetings where we all sat together and worked on code/did pair programming, as we would encounter issues that would block progress, and could only wait for a slow response on discord/in the next meeting, if we coded together as a group it would have been much easier to get the help we needed.

We recommend using some established front-end framework like React or Svelte rather than vanilla javascript like we used, as it would save a lot of time and effort.

Issues

An issue we have encountered and is ongoing is needing to spend time in planning how the game’s systems worked since the creators' report was a bit too ambiguous. The creators could be contacted and asked for clarification to resolve this.

Features in the waiting room include the: tutorial mode, region system, personnel and upgrading factories feature, the implementation of saving and loading data via a database, and viewing high scores. The database can be done by using a relational database, and the serializing of the gameState can help in adding the saving/loading feature.