Factory Manager Project Final Report

Image Source:
https://www.istockphoto.com/photo/young-master-in-hardhat-and-bearded-engineer-discussing-technical-sketch-gm1207928621-348960817

Prepared by
Luke Repta, Mike Apreza, Dongxin Zhang, Tao Wang

CS 440
at the
University of Illinois Chicago

April 2022

Table of Contents

List of Figures

I Project Description

1 Project Overview

Factory Manager is a business simulation web game where a player, taking the role of an entrepreneur, must manage and expand their empire of factories. At the beginning of a game, in the starting region, the player is given a startup fund and ability to build a factory for free. Using their startup resources, they have to manage their resources carefully in order to expand their influence and earn revenue. Each factory has their own set of machines and product they manufacture at the time. Each region has their own delivery system where the player can sell products to stores to earn money and/or pick up resources from another factory to build more products. A player cannot manufacture products without resources, and cannot earn money without selling products. Each region contributes to the player's overall revenue. In order to win, the player has to have at least one million dollars at the time where they also earn ten thousand dollar per unit of time. A player loses if they start earning a negative net-worth over a certain period of time. Their final score is determined by how much money they earn and the time it took them to win.

2 Project Domain

This project's purpose is to create a web game that simulates the business management of a company composed of multiple regions and factories. The project originally was created to implement the traveling knapsack problem, in which the goal is to find the shortest route to a certain destination while optimizing the value of items you carry where each item has a specific weight and value.

The features that are important to mention for testing purposes and/or to determine the product's value and/or correctness are as described below:

Welcome Page: Upon the execution of the program, the user can select to start a new game with or without the tutorial. If the checkbox for "Do Tutorial" is selected, then a message box will appear while many buttons are disabled to guide the player in a strict way.

Research System: The player is able to research nodes that require time and money in order to unlock different kinds of Machines, Products, and Bonuses. The nodes may have a prerequisite node that needs to be researched before and/or a node that becomes unlocked after the successful research of the current one. Researched nodes become disabled and unlocked nodes will cause an event to occur upon the selection of them. After a valid node is clicked, a bar at the top of the page will visually display how much time is left until the node has been successfully researched. It also shows what is being researched and once it is completed.

Supplies Button: Upon the selection of the button, a pop-up window displays the amount of Products and Resources the user has. It also shows which Factories the player has bought and how much money the player has.

Delivery System: The player can decide to sell Products and/or buy Resources. They can select up to three of those Products and Resources for a total of six items. After inputting the type and quantity of those items, the player can send out a truck to make the delivery. This would remove the money according to the Resources the user wants to buy, and/or the amount of Products the user wants to sell. Upon the selling of Products that the player has in stock, the value of money should go up when the delivery has been completed. On the other hand, upon the buying of Resources, the quantity of the specific Resources the user buys increases. The player can also see the time remaining for each truck to complete the delivery planned.

3 Relationship to Other Documents

This project is heavily based on the Factory Manager Project Report, as described by Agrawal et al. in [4], which contains all the details of the project including: a project overview, requirements and their acceptance tests, feature descriptions, design and class diagrams, scenario descriptions, and project issues. For section two of this document, the Factory Manager Tutorial Scenario, as described by Zhang et al. in [10], and Factory Manager Full Game Scenario, as described by Zhang et al. in [9],will be used, which contains goals and descriptions of the first two releases.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Resource: The fundamental items used to create Products, the parent class of Product, such that Resources can be used to create intermediate Products, which are then used as Resources to make more advanced Products. Can be bought at the “Resources” destination in the transportation system.

Product: Objects created by Machines in Factories by consuming resources (products are resources and as such can also be used as the resources for producing more advanced products). Can be sold at the “Products” destination in the delivery system, which is the primary means of making money in-game.

Machine: Produces the Factory selected Product type by consuming the Products required resources. The amount of Products produced depends on the type of Machine. Machines exist in an associated Factory, and consume the resources of the Region to which the Factory belongs.

Factory: An aggregation of Machines that is used as an interface between the Interval and Production Systems. A Factory has a certain allowed capacity for Machines, and performs the production calculation based on the amount and type of Machines present within. All Resources consumed come from, and all Products produced go to the Region to which the Factory belongs.

Region: An aggregation of Factories, Products/Resources, and the Transportation System for that Region. Has an allowed capacity of Factories, and stores all the Products and Resources that the Factories within the Region consume/produce.

Products can be sold and Resources bought via the Region's Transportation System, which is broken down into the Region's Trucks and Delivery Routes.

Truck: The Basic Object that performs the functions of the Transportation System. Has an allowed capacity, and can be loaded with up to three types of Products/Resources. Trucks go on Delivery Routes as specified by the user.

Delivery Route: A Delivery Route consists of a destination, cargo, duration, and Truck. The user may choose the destination between the Store for selling products, or the Resources destination for buying additional resources. After selecting the destination, the user chooses the cargo to be sold/bought, the quantity of that cargo to be sold/bought, and then confirms the route. Based on the cargo and destination chosen, the Delivery Route will have a specific calculated duration.

Research Node: Nodes that unlock additional Products/Machines/Bonuses once the user finishes researching them. Researching a node takes a designated amount of money and node specific required units of time to be completed.

Product Node: Subset of Research Nodes that unlock a new type of Product when researched. Besides the starter Product, Steel, all additional Product types must be unlocked via researching the corresponding Product Node.

Machine Node: Subset of Research Nodes that unlock a new type of Machine when researched. Besides the starter Machine, Basic Machine, all additional Machine types must be unlocked via researching the corresponding Machine Node

Bonus Node: Subset of Research Nodes that give a bonus modifier when researched (such as increasing the output efficiency of production in factories). All modifiers have a base value, and Bonus Nodes increase/decrease them based on the type of modifier.

More terms are seen in the Glossary (page 38).

4b UML and Other Notation Used in This Document

This document generally follows the version 3.0 OMG UML standard, as described by Fowler in [8]. Any exceptions are noted where used.

4c Data Dictionary for Any Included Models

gameState: The main Data Structure that stores all the data about the currently running game. The gameState has an array of all Regions, as well as several Maps used to contain the Products/Resources/Machines/Research Nodes in a unified format where the key of a Machine Research Node is the same as the key for the corresponding Machine that it unlocks in the Machine Map. Researched Nodes are stored in a Set.

ReactiveData: A generic object included in some of the other structures. Because vanilla Javascript does not natively support reactivity, this is how we enabled reactive

binding between UI elements and the corresponding objects. The ReactiveData object has a series of properties, and stores a list of "listeners" for those properties. The Getter/Setter properties of the ReactiveData object are overwritten such that modifications to a property "notify" all the listeners of that property (i.e. UI elements displaying some data), and when notified those listeners can call some function/execute some lines of code to update itself (like the Money UI element changing after the user spends cash on something).

Truck: Has the information of a truck entity, which tells the state of it. Besides having an identification number, it has variables for the current weight, time, and maps of the products and resources it carries. The weight is used to calculate the delivery time. The time variable has the number of seconds the truck still has left in a delivery. If the value is zero, the truck is ready for use. The map of products and resources are in the form of (key, value) where the key is the name of the product/resource and the value being the quantity of that item to be sold or bought, respectively.

Region: Contains maps for the player's total number of products and resources in the form of: (product type, quantity). Also contains an array of factories and a set of trucks that the player currently has. It makes all of the above reactive.

The key way we synchronize between the different data structures (like the Products/Resources/Machines/Research Nodes maps) is by having related objects correspond to the same key in their respective map. So a Research node that unlocks the new product Stainless Steel will have the same key in their two different maps (Research Node and Product maps). This allows for easy and performant synchronization between the two data structures for the Research system.

Production Formula: Products Produced = (numMachines * averageMachineLevel) * (throughputEfficiency) * (outputEfficiency) , as constrained by the number of possible products

Resources Consumed Formula: Resources Consumed = (numMachines * averageMachineLevel) * (throughputEfficiency) * (inputEfficiency) * (numResourcePerProduct)

II Project Deliverables

The game can be broken down to two modes; tutorial and full game mode. The tutorial mode has restrictions on what the player can interact with and messages to guide the player as well as informing them of various features. The full game mode also gives the player a fund and free factory. Unlike the first mode, it has no such restrictions.

1 First Release

The date of the first release is February 25, 2022.

As of the first release, we had the welcome page written in HTML with three factory buttons, a tutorial message box, and a list of side buttons where only the “Machines” and “Manage Deliveries” buttons worked. Once users select the first factory following the tutorial, they are able to click the “Machine” button to go to the inner factory page. There, eight machine buttons were placed where only one could be selected at a time. On a selected machine, which would be highlighted in red, there were two dropdowns for users to select a machine type and blueprint. Upon the selection of both, a generic product variable was incremented every second and reflected in the header. On the same page, on the right, there was a list of side buttons. The “Go Back” button took the player back to the initial game page. The “Research Products” button took the user to the research page. In the research product page, there was a table of buttons representing research nodes that unlocked different kinds of guitars. Users were only able to click on the first node of the first column to make a blueprint. Below the research table, if a node was being researched, the time remaining was shown. Only when the time is out, the node was disabled and its name was added to the blueprint dropdown in the machine page. Also, the node below it was able to be researched. Upon the complete research of the whole row, the next row was unlocked. A screenshot of each page that had some functionality beyond the “Go Back” button can be seen in Figures 1, 2, and 3 below.

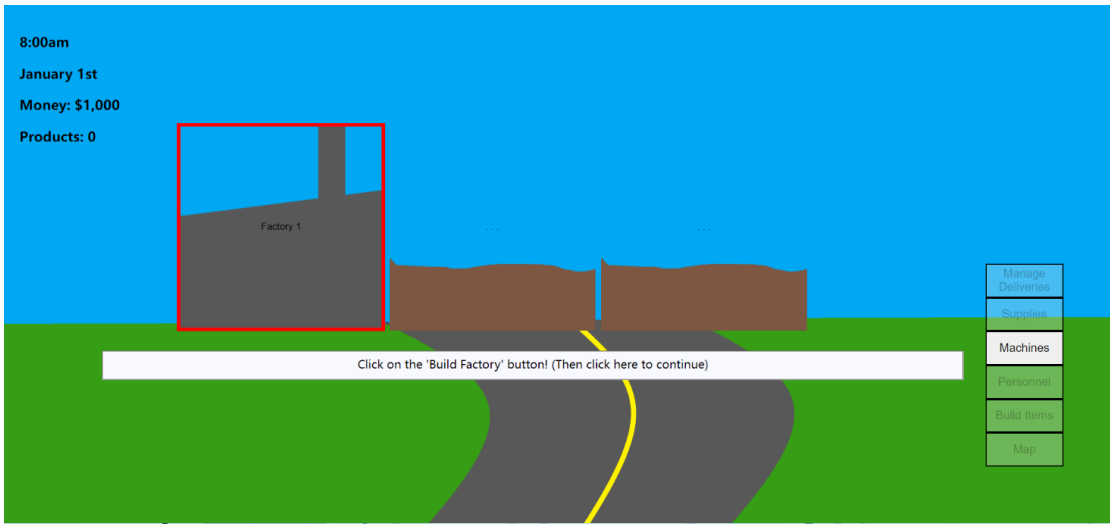

Figure 1 - Release 1 Welcome Page

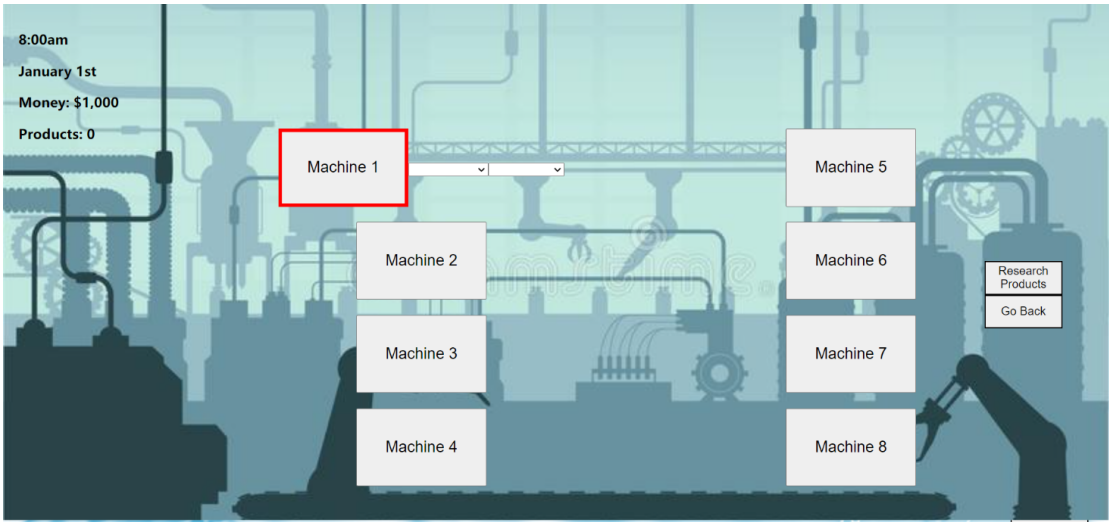

Figure 2 - Release 1 Machine Page

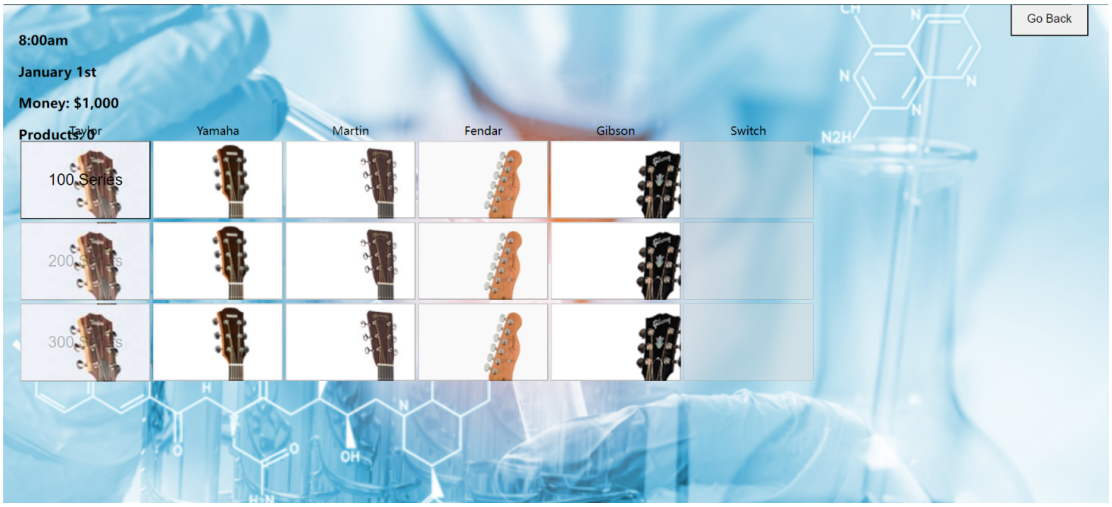

Figure 3 - Release 1 Research Page

Visually, everything was working, however, the logic behind it had not been implemented yet. Some classes were built to store data such as the selection of blueprints in a specific machine. There were also prototype lists of machines, research nodes, products, and resources. The current research nodes, factories, and machines were hard-coded into their respective pages.

2 Second Release

The date of the second release is on April 1, 2022.

As of the second release, there was a more appropriate welcome page with a “New Game” button followed by a checkbox that made users be able to switch between a normal game and tutorial mode. The welcome page can be seen in Figure 4 below.

Figure 4 - Release 2 Welcome Page

In the tutorial mode, the game worked almost the same as before in release one, however, the page had more functionality including the ability to build more factories when the right amount of money was acquired and more that will be mentioned later. Users could still click on the message box to see a new guide message that helped with becoming familiar with the game.

In the normal game page, the first factory is marked as default and users are able to build at no cost. The other two factories cannot be built unless the player has enough money as marked. In the header, it can be seen that time actually passed by. The money is also changed upon the spending of it. The side buttons are different as in release one; All of them are enabled besides the "Machines" one which can only work upon the selection of a built factory. However, the "Supplies" and "Map" buttons do not have event listeners. This page can be seen in Figure 5 below.

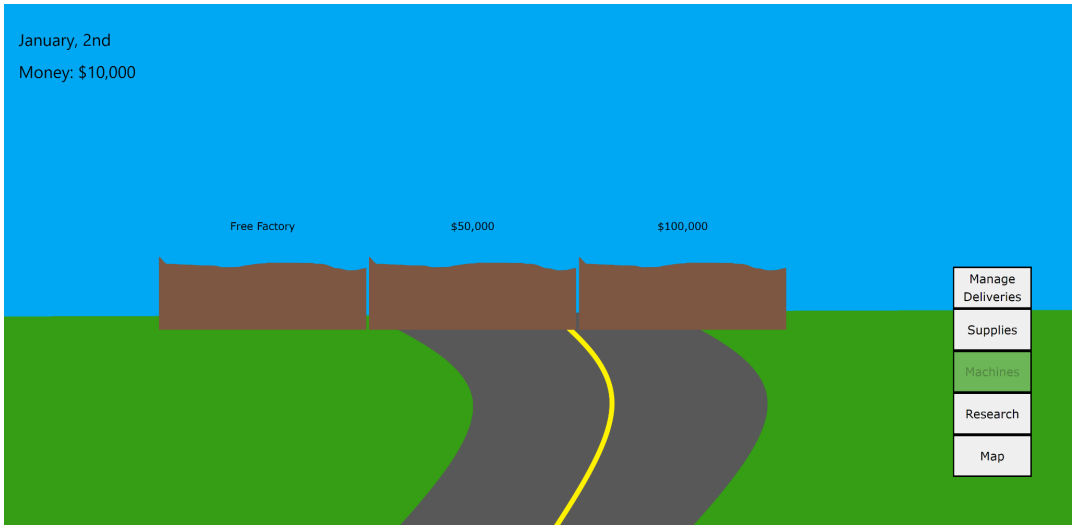

Figure 5 - Release 2 Normal Game Page

The research product page had been moved to the normal game page and tied to the "Research" button. After clicking it, the page loads with the same header as the game page and a dynamically loaded technology research tree as defined in researchList.js with the research nodes that includes its prerequisite and successor as seen in Figure 6 below. Each node is tied to a specific machine, product, or bonus object that contains necessary information such as the cost for a machine, weight of a product, or bonus percentage of a bonus node. Therefore, each node can only be researched upon the previous completion of the prerequisite as well as having enough money that is required for that node. If a product or machine is researched, after the time is counted down, the machine/product is added to its respective dropdown in the machine page.

```
function generateResearch() {
    window.gameState.researchTree = new Map([

        ["steel" , new ResearchNode({
            id : "steel",
            name : "Steel Production",
            cost : 2000,
            time : 20, // seconds
            type : "product",
            requiredPrereqNodes : null,
            nextNodes : ["stainlessSteel"]
        })],

        ["stainlessSteel" , new ResearchNode({
            id : "stainlessSteel",
            name : "Stainless Steel Production",
            cost : 2500,
            time : 25,
            type : "product",
            requiredPrereqNodes : ["steel"],
            nextNodes : ["advancedAlloy"]
        })],

        ["advancedAlloy" , new ResearchNode({
            id : "advancedAlloy",
            name : "Advanced Alloy Production",
            cost : 3500,
```

Figure 6 - researchList.js File

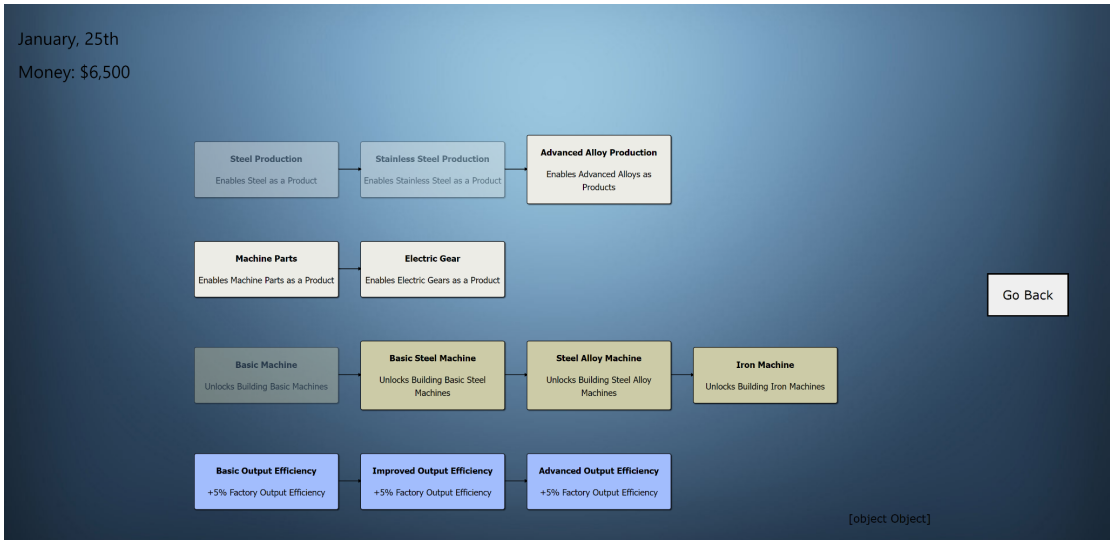

Figure 7 - Release 2 Research Page

The machine page has only a single side button to take the user back to the game page. The two dropdowns that used to be near the selected machine in release one were deleted. In replacement, a single drop down for the product the user wants to build was added to the top of the machine page. Each machine has a single dropdown where the user can select the type of machine they want to use, which costs money depending on the machine type. Upon the selection of a blueprint (product the user wants to build) and machine type, products begin to become manufactured as seen in a small area near the lower left corner, which displays how many units the user has of a product and resource, which is added for testing purposes. If the blueprint is ever changed due to more products having been researched, then the information of the product and resource is updated near the lower left corner. As products are produced, the value is saved in the game.

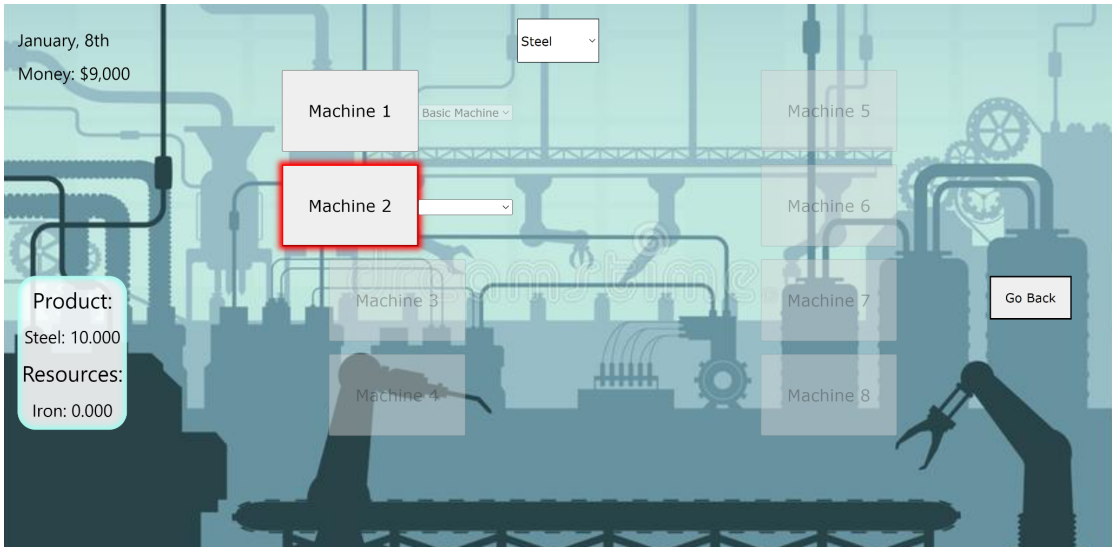

Figure 8 - Release 2 Machine Page

In the plan deliveries page, which is tied to the “Manage Deliveries" button in the game page, the users can click on the “Stores” or “Resources” buttons, which display a modal asking a player to select a product/resource type and quantity they wish to sell/buy. Each modal has a limit of three product/resource types and quantities. The product text boxes next to the product type dropdowns have a minimum value of zero and maximum value of the number of that product the user has. For the resource textboxes, the minimum is the same, but the maximum is unlimited. If the player selects “Done” in any of the modals, a message is displayed that the first numerical input is required if the player missed it. If any other product types are selected, the numerical input becomes required. Clicking on “Cancel” or the close button closes the modal. If all is selected appropriately, upon the selection of the “Done” button, the welcome page will show, restarting the game. Although the “Deliver” and “See Deliveries” buttons are disabled, there are event listeners attached to them. The first button causes nothing to occur. The second button only opens a blank modal, which will be used to display the truck currently in use, its ID number, how much time is left for that delivery, and a cancel button if the player wishes to stop the delivery. However, this could not be tested in time. The plan deliveries page with a product selection modal can be seen in Figure 9 as seen below.

Figure 9 - Release 2 Plan Deliveries Page

In terms of the backend, plenty of other classes were added to support multiple regions (Region class), deliveries (Truck class), money modification (Money class), and date change (GameDate class). Every page and item are now dynamically generated via JavaScript. An interval file was created to deal with all the issues that are associated with time such as deliveries, date change, researching, and product manufacture. There is also a winning condition included where if the player has one million dollars and an income of ten thousand dollars per time unit, an alert is shown congratulating the player and informing them that they can start a new game or continue playing in the current one. There was also a reactive file created. This file handles all real-time updates. Additionally, a GameState class was created to hold all of the information a user needs in every region including: factories, researched nodes,

money, the current node being researched, the time left of that node (if any), regions, pages, research nodes, machines, products, and resources,

3 Comparison with Original Project Design Document

We designed our user interface based on Figure 11 of the Factory Manager Final Report (55 Agrawal et al.). There are a few differences; First, the date we display is only the month and day while the figure includes the time. This was done for simplicity's sake. Second, this project excludes the "Personnel" and "Build Items" buttons. Since this project does not include any functionality for the "Personnel" button, we have removed it. As for the "Build Items," the original report does not include any description as to what the button was to be used for, therefore, we have substituted it for the "Research" button, which shows the technology research tree. Since the original report does not describe anything about the other visual elements, the team decided to make the UI simple and easy to use.

In terms of classes, for the Factory class, which is defined in Figure 16 in the original report (58 Agrawal et al.), we have it completely different compared to the original design. Instead of a factoryName, we have an index to separate factories. As for the products, there is only one product type in our version as opposed to multiple per factory. Both versions do have an array of machines. We do have a Product class, however, it extends our Resources class, which is not present in the report. Our Product class does have a name, but that is the only similarity present. The Machine class in this version of the project has everything present in the design document and more, but it does not have a usableBlueprints because our factories work differently. In terms of the managing of deliveries feature, we do have a Truck class, but it is very different from the original design seen in Figure 17 (59 Agrawal et al.). Additionally, the prototype has a Region class that supports the possibility of having different regions with their own set of factories, machines, and such as defined in the "Move to Different Region Scenario" (15 Agrawal et al.). Unlike the original design, we have created a GameState class that stores the state of the game such that the player can switch pages without losing data. There is also a file dedicated to deal with all the timing scenarios such as researching nodes, deliveries, and checking the winning condition.

As for functionality, a good amount of variables that both versions share are not used in our version. In the design report, the functionalities of the machines, factories, and delivery system are more complicated (14-15 Agrawal et al.). For example, our machines in the whole factory only support the production of one type of product at a time and machine slots must be bought. Also, our factories are not upgradable, instead you research and buy higher quality machines to replace existing ones.

Our planning delivery system is very simplified. Section 4c of the Factory Manager Final Report (14-15 Agrawal et al.) explains that delivery trucks are to be managed per factory, whereas our prototype instead uses a common set of delivery trucks across an entire region. We made this decision to facilitate the ease of use of the delivery system, as well as to make regions more cohesive, such that it is possible to have a factory producing a low level product, which is used as a resource in another

factory in that region producing a more advanced product. Our prototype did however maintain a very similar design in the delivery system, with trucks having a set capacity of cargo they can carry, which is either used to sell finished products at the "stores," or to buy resources at the "resource location". A lot of our work revolved around not just implementing a similar system to the one proposed in the original design report, but more importantly exposing this information to the user in an intuitive and easy to use interface, and this was another area where having trucks be used region-wide made things simpler for the user to use and understand.

On the other hand, our research tree is more complicated because we include the feature of being able to research output efficiencies and machines whereas the original document talks about only being able to research blueprints/products. Our research tree is also more user friendly, as it displays a helpful tooltip showing the cost of the node, how long it will take to research, and information about the product/machine/bonus it unlocks.

III Testing

1 Items to be Tested

The code that is to be tested is written individually by three of the four group members. There is a member that did not write code, therefore, code could not be tested for that fourth member.

It is also to be noted that the delivery system and supplies modal work together. To test the research and delivery systems, the Machines page is relied on to be accurate.

The items to be tested themselves can be summarized and seen as follows:

a Member 1

File Name: researchSystem.js

Author: Luke Repta

Date: 4/20/22

Description: Contains most of the logic for the research system, as well as the construction for some of the user interface (like the info tooltip for each node). Handles researching nodes and unlocking their bonuses, alongside animating the research progress bar.

Link: researchSystem.js

b Member 2

File Name: deliverySubsystem.js

Author: Mike Apreza

Date: 4/20/2022

Description: The file contains the function to create and return the modal that appears upon the click of the "Stores" button. There are two functions that are tied to it, which are called upon when the "Done" button is clicked and checks if the user input on the modal is valid. It also has the function that returns the modal when the "See Deliveries" button is clicked. The last function is executed when the "Deliver" button is selected.

Link: deliverySubsystem.js

c Member 3

File Name: WelcomePage_SuppliesModal.js

Author: Dongxin Zhang

Date: 4/20/2022

Description: Contains most of the functions for the supplies modal and welcome page, which creates the page and the inner logic to display the status of factories, products, and resources.

Link: WelcomePage_SuppliesModal.js

2 Test Specifications

BAR1 - Boundary of the Amount of Resource

Description: Test the boundary of buying resource in store

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Functionality.

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- NG1 - New Game Button
- The Plan Delivery page must be completed

Test Procedures: The tester will start a new game, then click on the Mange Deliveries button. There, the tester should click on the Resources button. The tester should type in an excessively large number and click 'Done'.

Input Specification: Tester only needs to launch the game and go to the resources modal. The tester should type in the amount of resources. and click on the 'Done' button.

Output Specifications: There should be a pop-up window showing that the amount is invalid.

Pass/Fail Criteria: The test will pass if there is a pop-up window showing that the input value is not valid.

DSM1 - Display the Stores Modal

Description: When the Stores button is clicked, the modal should display.

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Visual Quality and Functionality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: NG1 New game test and the Manage Delivery page must be completed successfully for the Stores modal to be tested.

- NG1 - New Game But
- The Plan Delivery page must be completed

Test Procedures: The tester will start a new game, then click on the Manage Deliveries button. There, the tester should click on the Stores button.

Input Specification: Tester only needs to launch the game, then click the new game button, and immediately click on the Manage Deliveries button, then the Stores button, no special input to the program is necessary.

Output Specifications: Upon clicking the Stores button, the stores modal should be displayed.

Pass/Fail Criteria: Once the tester clicks on the Stores button, there should be a pop-up window in which there is a message saying what users can do within the window. There should be three dropdowns for the users to choose at most three products and 3 boxes for users to put the amount they want to sell. At the bottom of the window, there should be two buttons named 'Done' and 'Cancel' so that users are able to Confirm the amount they want, or to cancel what they put in and close the window. If the answers of all those mentioned above are YES, then Pass. Otherwise, Fail.

CI1 - Check Input

Description: When there are only one type or two of products, the dropdowns of product2 and product3 are useless. Test if the two dropdowns would be marks unavailable in some ways

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Visual Quality and Functionality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- NG1 - New Game But
- DSM1- Display the Stores Modal
- The Manage Delivery page must be completed

Test Procedures: Tester needs to launch the game, go to the Manage Delivery page, then click on the Stores button. Tester checks if the dropdown shows NA when there is only one type of product. Then the tester should research another type of product and see if the NA is removed.

Input Specification: Tester needs to launch the game, go to the Manage Delivery page, click on the Stores button. In order to get a new product, the tester should buy some resources and go to the machine page to make products.

Output Specifications: Upon clicking the Stores button, there should be NA on the dropdown of product 2 and 3 if there is only one product.

Pass/Fail Criteria: When there is only one type of product, the dropdowns of product2 & 3 should be placed NA. If there are two types, the product3 should be placed in NA. If there are three types or more than three types, all dropdowns should work. If the answer of all those mentioned above are YES, then Pass. Otherwise, Fail.

ED1 - Enable Delivery Button

Description: After products and amount are selected, the deliver button should be able to be clicked on.

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- NG1 - New Game But
- The Manage Delivery page must be completed

- The implementation of Resources modal must be completed

Test Procedures: The tester will start a new game, then click on the Manage Deliveries button. There, the tester should click on the Stores button, select resources and type in amount, and hit the 'Done' button.

Input Specification: Tester needs to launch the game, go to the Manage Delivery page, click on the Stores button. The tester is supposed to select resources and provide an amount, then click the 'Done' button.

Output Specifications: Upon clicking the Done button, the modal should be closed and the Delieve button should be able to click on.

Pass/Fail Criteria: The 'Delivery' button should not be grayed out, this signifies a failure.

VT1 - Visualize the Timing of Trucks

Description: SeeDelieveries is a modal that shows the minute left of a truck.

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- NG1 - New Game Button
- ED1 - Enable Delivery Button
- The Plan Delivery page must be completed

Test Procedures: Click on the resource button and buy some, then click on the Deliver button, and finally click on the See Deliveries button.

Input Specification: Tester needs to launch the game, go to the Manage Delivery page, click on the Resource button and buy some resources and deliver it.

Output Specifications: Upon clicking the SeeDelivery button, there should be a window showing a countdown of a truck.

Pass/Fail Criteria: There should be a countdown showing the minute left for a truck to deliver the resource. If yes, then pass. Otherwise, fail.

DR1 - Deliver Resource

Description: The deliver function calculates the time necessary to deliver products and resources. It removes money from the player if resources are bought.

Items covered by this test: deliverySubsystem.js

Requirements addressed by this test: Visual Quality and Functionality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- NG1 - New Game Button
- ED1 - Enable Delivery Button
- VT1 - Visualize the timing of trucks
- The Plan Delivery page must be completed

Test Procedures: Go to the Resources' modal, select resources and type in amount, and hit the 'Done' button. The tester should click on the delivery button, then go to the previous page and click on the supplies modal.

Input Specification: Tester needs to launch the game, go to the Manage Delivery page, and buy some resources. Then the tester goes to the supplies modal and checks the number.

Output Specifications: Upon clicking the supplies button, the modal should show the correct amount of money and resource.

Pass/Fail Criteria: In the supplies modal, the final amount of each resource and money should be correct. If not, fail.

NG1 - New Game Button Starts New Game

Description: A new game should correctly be started when the new game button is clicked on the welcome page.

Items covered by this test: WelcomePage_SuppliesModal.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: N/A

Test Procedures: The tester should launch the program, then click the new game button.

Input Specification: Tester only needs to launch the game, then click the new game button, no special input to the program is necessary.

Output Specifications: Upon clicking the new game button, the main page should be displayed and a new game properly started.

Pass/Fail Criteria: The test will pass if upon clicking the new game button, the user is taken to the main page, indicating a new game has been successfully created.

SM1 - Supplies Modal Empty on New Game

Description: When a new game has started and before the user has built any factories or acquired any resources/products. The modal showing the supplies should display 0 for all products and resources, and show that there are no factories.

Items covered by this test: WelcomePage_SuppliesModal.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: NG1 New game test must be completed successfully for the supplies button to be tested.

Test Procedures: The tester will start a new game, then immediately select the supplies button without doing anything else. The tester will then verify that the supplies modal accurately shows 0 products/resources and no factories built.

Input Specification: Tester should only start a new game then immediately click the supplies button.

Output Specifications: Upon clicking the supplies button, the supplies modal should be displayed with correct information (as specified in the test procedure).

Pass/Fail Criteria: The test will pass if the information displayed in the supplies modal is correct, i.e. 0 products/resources, and no factories built.

SM2 - Supplies Modal show Factories Built

Description: The supplies modal should accurately display the status of each factory (whether it has been built or not).

Items covered by this test: WelcomePage_SuppliesModal.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: NG1 New game test must be completed successfully for the supplies button to be tested.

Test Procedures: The tester will start a new game, then immediately select the supplies button without doing anything else. The tester will then verify that the

supplies modal accurately has no factories built. The tester should then buy another factory, then reopen the supplies modal and verify that the new factory is marked built, and repeat until all factories are built.

Input Specification: Tester should only start a new game then immediately click the supplies button. No special input to the program is necessary, the tester should follow the procedure outlined above.

Output Specifications: Upon clicking the supplies button, the supplies modal should be displayed with correct information (as specified in the test procedure).

Pass/Fail Criteria: The test will pass if the information displayed in the supplies modal is correct, i.e. for each built factory, that factory is marked as such in the modal.

SM3 - Supplies Modal shows Products/Resources

Description: The supplies modal should accurately display the amount of each type of product/resource.

Items covered by this test: WelcomePage_SuppliesModal.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: DR1 Resource Buying works

Test Procedures: The tester will start a new game, then build one factory. The tester should use the delivery system to buy 12 Iron, and 5 Nickel. The tester should then build one machine in the built factory, and wait for it to convert all of the 12 Iron into 4 Steel. They will then test that upon clicking the supplies button, the supplies modal accurately shows that the region has 4 Steel, and 5 Nickel, and 0 for every other product/resource.

Input Specification: Tester should only start a new game. No special input to the program is necessary, the tester should follow the procedure outlined above.

Output Specifications: Upon clicking the supplies button, the supplies modal should be displayed with correct information for products and resources (as specified in the test procedure).

Pass/Fail Criteria: The test will pass if the information displayed in the supplies modal is correct, i.e. 4 Steel and 5 Nickel, and 0 for every other product/resource is displayed.

SM4 - Supplies Modal shows Changing Products/Resources

Description: The supplies modal should accurately display the amount of each type of product/resource as they are changing.

Items covered by this test: WelcomePage_SuppliesModal.js

Requirements addressed by this test: Visual Quality

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: DR1 Resource Buying works

Test Procedures: The tester will start a new game, then build one factory. The tester should use the delivery system to buy 30 Iron. The tester should then build one machine in the built factory, and immediately go back to the main page and click the supplies button to see the supplies modal. While the supplies modal is open, the tester should be able to see the iron going down in steps of 3, while the number of Steel is incremented in steps of 1 (i.e. shows Iron being converted into Steel) 30 Iron and 0 Steel should change over time into 0 Iron and 10 Steel. They will then test that the supplies modal accurately shows that the region has 10 Steel, 0 Iron, and 0 for every other product/resource.

Input Specification: Tester should only start a new game. No special input to the program is necessary, the tester should follow the procedure outlined above.

Output Specifications: Upon clicking the supplies button, the supplies modal should be displayed with correct information for products and resources (as specified in the test procedure).

Pass/Fail Criteria: The test will pass if the information displayed in the supplies modal is correct, i.e. 30 Iron and 0 Steel gets converted over time into 0 Iron and 10 Steel, and 0 for every other product/resource over time.

GRD1 - Generate Research Page Dynamically

Description: Upon any list of research nodes, they should appear as specified in the research page.

Items covered by this test: generateResearchPage() from researchSystem.js.

Requirements addressed by this test: Visual Quality.

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: N/A

Test Procedures: Click on the “Research” button of the initial game page. Hover the mouse over every node to see the on-hover action. Make sure all the text is what appears in the file that has all the information for the screen. Ensure the

header is still visible, the background is the image specified in the function, there is a “Go Back” button, and there is a progress bar at the top center.

Input Specification: The file that contains the research list, which has all the unique information pertaining to each node. In this case, it is called researchList.js.

Output Specifications:

- Upon the clicking of the “Research” button of the initial game page, there should be a line of nodes where parents point to their children.
- The header should still be there showing the month, day, and money amount.
- There is a “Go Back” button to the right.
- The background should be the image specified in the function.
- The progress bar should be at the top center.
- A line of nodes should be one unique color.
- Each node should have a title and description.
- Upon having the mouse hover over the node, it should display information about how much the node costs and time it takes to research.
- Product nodes should also display what Resources are needed to build the Product and how much a single unit would sell for.
- Machines nodes should also show the cost of the Machine itself and how much it boosts production.
- Bonus nodes also show a small description.

Pass/Fail Criteria: The test will pass if the node’s content and on-hover element contains only the information seen in the researchList.js. Also, the nodes should be in a certain order depending on their prerequisites.

GB1 - Go Back Button

Description: There should be a “Go Back” button in the Research page that changes the page to the initial game page without resetting the game.

Items covered by this test: goBackMain() from researchSystem.js.

Requirements addressed by this test: Functionality.

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies:

- GRD1 - Generate Research Page Dynamically
- RN1 - Research Node(s)

Test Procedures: Build any number of factories available. Click on the “Research” button of the initial game page. Click the “Go Back” button. Go back

to the research page and research a node. Wait for it to finish, then go to the initial game page. Return to the research page.

Input Specification: Have enough money to build a factory and research a node.

Output Specifications:

- The number of built factories should not change.
- The number of nodes researched should not change unless clicked on.

Pass/Fail Criteria: The test will fail if the number of built factories and/or researched nodes changes without anyone clicking on them.

RN1 - Research Node(s)

Description: Clicking on a node with enough money will cause the node to be disabled as the progress bar shows the item being researched and time progressed as specified in the node before unlocking the specified item and any child node.

Items covered by this test: research(), completeResearch(node), startResearchInterval() and the if-statement before the startResearchInterval() function from researchSystem.js.

Requirements addressed by this test: Functionality and Visual Quality.

Environmental needs: No specific requirements, only need to run the game.

Intercase Dependencies: N/A

Test Procedures: In the research page, click on a node having enough money. After it has been researched, click on another. Ensure there is not enough money before trying to research another node.

Input Specification: Have enough money to research more than one node. Run out of money before researching the last node for the test.

Output Specifications:

- Only the researched nodes should be disabled.
- The progress bar should display the last researched node (if any).
- Researched Products should be reflected in the blueprint dropdown of the Machines page.
- Researched Machines should be reflected in the dropdowns by a selected Machine.
- Research Bonuses should make the product manufacture faster.

Pass/Fail Criteria: The test will fail if any non-researched nodes are disabled. The test will pass when the progress bar only works as described, only researched nodes are disabled, and unlocked items/bonuses function as intended.

3 Test Results

BAR1 - Boundary of the Amount of Resources

Date(s) of Execution: 04/23/2022

Staff conducting tests: Dongxin Zhang

Expected Results: There should be a pop-up window showing that the given amount is not valid.

Actual Results: There is a pop-up window showing that the given amount is not valid.

Test Status: Pass

DSM1 - Display the Stores Modal

Date(s) of Execution: 04/22/2022

Staff conducting tests: Dongxin Zhang

Expected Results: Once the "Stores" button is clicked, there should be a pop-up window. there should be a message saying what users can do within the window. There should be three dropdowns for the users to choose at most three products and three boxes for users to put the amount they want to sell. At the bottom of the window, there should be two buttons named 'Done' and 'Cancel' so that users are able to Confirm the amount they want or to cancel what they put in and close the window.

Actual Results: The result satisfy the requirements mentioned above

Test Status: Pass

ED1 - Enable Delivery Button

Date(s) of Execution: 04/22/2022

Staff conducting tests: Dongxin Zhang

Expected Results: The 'Delivery' button should not be grayed out.

Actual Results: The 'Delivery' button is not grayed out.

Test Status: Pass

CI1 - Check Input

Date(s) of Execution: 04/22/2022

Staff conducting tests: Dongxin Zhang

Expected Results: When there is only one type of product, the dropdowns of product2 and product3 should be placed N/A. If there are two types, the product3 should be placed in N/A. If there are three types or more than three types, all dropdowns should work.

Actual Results: The result satisfies the requirements mentioned above.

Test Status: Pass

VT1 - Visualize the Timing of Trucks

Date(s) of Execution: 04/22/2022

Staff conducting tests: Dongxin Zhang

Expected Results: There should be a countdown showing the minute left for a truck to deliver the resource.

Actual Results: The result satisfies the requirements mentioned above.

Test Status: Pass

DR1 - Deliver Resource

Date(s) of Execution: 04/22/2022

Staff conducting tests: Dongxin Zhang

Expected Results: In the supplies modal, the final amount of each product should be correct.

Actual Results: The result satisfies the requirements mentioned above.

Test Status: Pass

NG1 - New Game Button Starts New Game

Date(s) of Execution: 04/22/2022

Staff conducting tests: Luke Repta

Expected Results: Upon clicking the new game button, the application should be on the main page, indicating that a new game has been successfully created.

Actual Results: After clicking on the new game button, the page changes from the welcome screen to the main page, showing that a new game has been created.

Test Status: Pass

SM1 - Supplies Modal Empty on New Game

Date(s) of Execution: 04/22/2022

Staff conducting tests: Luke Repta

Expected Results: Supplies Modal should display 0 for every product and resource, and there should be no check marks in the boxes representing the factories that have been built. Only the current amount of money should be shown.

Actual Results: The supplies modal accurately shows that the user has 0 products/resources, no factories built (no check marks), and displays the current amount of money.

Test Status: Pass

SM2 - Supplies Modal Shows Factories Built

Date(s) of Execution: 04/22/2022

Staff conducting tests: Luke Repta

Expected Results: Supplies Modal should display 0 for every product and resource, and for each factory that the user has created, the corresponding box representing that factory should have a checkmark.

Actual Results: The supplies modal accurately shows that the user has 0 products/resources, and correctly shows what factories have been built, regardless of the order they have been built.

Test Status: Pass

SM3 - Supplies Modal Shows Products/Resources

Date(s) of Execution: 04/22/2022

Staff conducting tests: Luke Repta

Expected Results: Supplies Modal should accurately display the amount of resources and products. Following the test procedure, this should be 5 Nickel, 4 Steel, and 0 for every other product/resource

Actual Results: The supplies modal accurately shows that there is 5 Nickel, 4 Steel, and 0 of every other type of product/resource.

Test Status: Pass

SM4 - Supplies Modal Shows Changing Products/Resources

Date(s) of Execution: 04/22/2022

Staff conducting tests: Luke Repta

Expected Results: Supplies Modal should accurately display the amount of resources and products, as the resources are being consumed and products created. The tester should be able to see the Iron decreasing by 3 at a time until reaching 0, at the same time as Steel is increasing 1 at a time until reaching 10.

Actual Results: The supplies modal only shows the number of products/resources accurate to the moment it is clicked, it does not update over time.

Test Status: Fail, supplies modal does not update when on screen, only when the button is clicked (i.e. resource/product counts become inaccurate if they are changing while the modal is open)

GRD1 - Generate Research Page Dynamically

Date(s) of Execution: 04/23/2022

Staff conducting tests: Mike Apreza

Expected Results: The Research page should have the background of researchBackground.jpg. Also, it should show four lines of nodes of unique colors where parents point to their children as specified in the researchList.js. The header should continue showing the month, day, money amount, and progress bar. The back button is at the right. Each node should have a title and description. Upon hovering the mouse over the node, it should display information about how much the node costs and time it takes to research as well as other information as specified in researchList.js.

Actual Results: All the information (i.e. title, description, hover information) is consistent in terms of formatting and specific text according to the node type and researchList.js.

Test Status: Pass.

GB1 - Go Back Button

Date(s) of Execution: 04/23/2022

Staff conducting tests: Mike Apreza

Expected Results: The number of factories built and researched nodes should not change unless the mouse has clicked on them. Money should be consistent in the header according to if there was any money spent at all.

Actual Results: Once a factory is built, it stays built no matter how many times the user goes back and forth between the initial game and research page. The same goes for the researched nodes.

Test Status: Passed; Once a factory is built and node is researched, they stay that way.

RN1 - Research Node(s)

Date(s) of Execution: 04/23/2022

Staff conducting tests: Mike Apreza

Expected Results: Only known researched nodes should be disabled, which need to be clicked unless they are the starter researched nodes. Nodes can only be researched when enough money is available. The progress bar should visually show how much time has passed (green) or is left (gray) as well as the node being researched. Once the time has passed, the bar should say that the node has been researched.

Researched Products should be available in the blueprint dropdown of the Machines page. Researched Machines should be available in the dropdowns by a selected Machine. Research Bonuses should make the product manufacturing process faster.

Actual Results: Researched nodes are disabled. Nodes currently being researched are also disabled and the time remaining is displayed in the progress bar. Also, if the nodes take longer to research, the bar is visually seen to progress slower. Nodes can only be researched when there is enough money, otherwise nothing happens. The progress bar shows nodes currently being researched and those that have already been.

Researched Products are always displayed in the blueprint dropdown of the Machines page. Researched Machines are always available in the dropdown by a selected machine. Researched Bonuses are added to the output efficiency; Using two hundred iron takes less time when there is a bonus compared to no bonus at all.

Test Status: Passed.

4 Regression Testing

This section is not applicable.

IV Inspection

1 Items to be Inspected

The code that is to be inspected is written individually by three of the four group members. Because one member that did not write code, code belonging to them could not be inspected. The items themselves seen as follows:

a Member 1

See section 3 subsection 1a.

b Member 2

See section 3 subsection 1b.

c Member 3

See section 3 subsection 1c.

2 Inspection Procedures

The inspection process is conducted in one in-person meeting. In the meeting, each person pulled up the code of one group member, which was selected by the member and copied to its own file. Then, the code was individually read through once before the rest of the group analyzed it together and discussed it based on certain criteria:

- Efficiency
    - Could it have been less repetitive?
    - Could it have been further broken up?
    - Is the data structure used efficient, and does it logically reflect the needs of the program?
    - Are steps taken to reduce the performance impact of complex or numerous calculations?
- Readability
    - Layout, does the code flow logically from one piece to the next?
    - Code style: Is the code understandable and logically structured?
    - Are descriptive variable and constant names used in accord with naming conventions?
    - Are there variables with confusingly similar names?
    - Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?
    - Spelling/grammatical errors in variables/comments?
    - Functions/class definitions not too long?
    - Magic numbers/unexplained constants?
    - Is each block of code well-commented?

The results were discussed in-person.

3 Inspection Results

a Member 1

- What was inspected: researchSystem.js
- Who did the inspection: Mike, Tao, and Dongxin
- The inspection occurred at 2:45 PM on April 20th, 2022.
- Results:
  - Code commenting could be more in depth. A lot of code is unexplained/has no comments.
  - More breaking down of the code into functions, in particular the very large for loop in the generateResearchPage() function should probably be broken down into at least two functions (like for example containing a break statement for the creation of the tooltips.
  - The efficiency seems good, there is some repetition in the for (let node of nodes) loops, which could be combined into a single loop, it was said this was for style reasons, but this could be avoided if the code was more functionalized (i.e. createTooltip() can replace the entire large block of code to add a tooltip individually, and the code to create custom css styling for each node can also be broken down into its own function to prevent the generate research page from stretching into 160+ lines)
  - Spelling/grammar seems fine.
  - Magic number 124 in the startResearchInterval() function, brief explanation given in the code, but it is not really sufficient to explain why the exact number 124 is used rather than something like 125 or 123?
  - Variable/function names are descriptive enough, again more functions could also be used to make the code more readable.
  - Could have moved some styling into a css class.

b Member 2

- What was inspected: deliverySubsystem.js
- Who did the inspection: Luke, Tao, and Dongxin
- The inspection occurred at 3:10 PM on April 20th, 2022.
- Result:
  - The comments that are present seem good at summarizing the code below, but more comments could have been included too.
  - Lots of lines are long and go off the screen, which could have been made into two lines to keep it readable.
  - A lot of the code in the “stores” and “submitProducts” functions could have been merged into loops since a lot of code is repetitive.
  - Some variable names could have been more descriptive.
  - Could have saved some space and time by calling the getElementByID function once per function and using it as needed.

- Since truckReady is equal to -1 is checked before, seeing if that's the case is redundant in the submitProducts function.
- Since the data structure for containing trucks is a Set, it was necessary to iterate through all the trucks instead of being able to access a single truck one way.
- The "alert" does not always work in some browsers. It also freezes the application, which is something that would be avoided. Something else could have been used.
- Could have deleted the last few lines of code in the deliver function since it's added in another function.
- Some functions are a bit over one hundred lines, but putting some repetitive code into a loop could have prevented this.
- Could have moved some styling into a css class.

c Member 3

- What was inspected: WelcomePage_SuppliesModal.js
- Who did the inspection: Mike, Tao, and Luke
- The inspection occurred at 3:44 PM on April 20th, 2022.
- Results:
  - Instead of for(i of …) for(const [productId, product], it is iterating a map, so [productId, product] lets the reader know that you are getting the key, value and using those, rather than i[0] and i[1], which makes you think that it is an array of some sort.
  - Use a single variable for document. getElementByID() (get it once and store it in a variable, rather than calling getElementById() many times)
  - There could be a underscore between two words of a variable, or camelcase could be used, rather than none: suppliesmodaldiv → suppliesModalDiv or supplies_modal_div
  - A few more comments should be added, so that others can more quickly understand what each part is doing
  - Could create objects in one liner instead of giving ID in the second line. Or if it is not going to be created in one line, be consistent with that and not switch between one liners with Object.assign and doing it manually property by property
  - Shorter ID name, since we logically know that some div in the suppliesModal function has to do with the supplies modal

V Recommendations and Conclusions

For the testing, the only failing test was: SM4 - Supplies Modal Shows Changing Products/Resources. Further testing/inspection is not needed, instead the feature must be implemented. As it stands now, the UI elements are created when the modal is shown, so a possible implementation to resolve the failing test case would be to use the same system as the reactive product/resource counts in the machine page. Setting those UI elements to listen to their respective product/resource property in the

gameState, then defining an (arrow) function to update their text content would be sufficient. The checkboxes showing what factories are built do not need to be changed, as the player cannot change how many factories are built when in the modal regardless. If this feature were to be implemented, we would do regression testing on the modal to see if it still passes all the previous test cases, and that it now passes test case SM4.

As for the inspection, all the files of code had some similar flaws. Everyone could have added more comments in terms of quantity and quality. This could be especially helpful where there are places that have unexplained number literals. Also, everyone had an issue with readability; Many lines of code could have been moved to their own functions, broken up into two lines, or moved into being a part of a different file. Variables in the program could have been more descriptive. Something to note is that everyone had their own way of doing that. One of the main flaws is that the code could have been more efficient by simply merging two loops or calling a function that retrieves something only once. One person had issues with redundancy. To improve such flaws, it can be as simple as going back to move, delete, or merge code. However, to prevent this from being done at this great of a scale in the future, it can be mandatory to go back and review code to improve efficiency. Variables can also be changed, especially with tools available that can search up a word in all the files and replace it with another. To make the program more efficient, it can be helpful to have another inspection process again after the first modification.

VI Project Issues

1 Open Issues

One of the issues that we have struggled with throughout the creation of the prototype is the real intention of the creators regarding how the game's systems work exactly. A lot of the report was ambiguous, therefore, the team had to do a bit more planning to make sure that the game was actually entertaining to some degree. This issue could be resolved by contacting the creators and asking for further specification on their intentions, which could even be that the developers fill in the gaps. However, this cannot be determined without speaking to them.

2 Waiting Room

In terms of improving features that are somewhat implemented, the tutorial mode can be continued further. As of now, the guidance stops once the user clicks on the "Machines" button. The tutorial needs to still guide the player through the product manufacturing process, research tree, buying resources, selling products, noting how the region system works, and how to win.

One of the requirements that have been set up is the region system. In the current version of the game, only one region is supported. Once all the other game features have been tested, it is just a matter of replicating the region.

Another part of the factories in the game is adding “personnel” to “work” on the machines, which would make the game a bit more complex. Regarding the factories, another feature would be the ability to upgrade the factories, which would require planning as to how it would benefit the player when done so.

One of the most important features yet to be added is player data being saved into a database. As of now, all or at least most of the important data is saved in the gameState object. Similarly, the loading of that data would also need to be implemented.

The saving and loading of data would also require the log-in feature to be implemented.

The original Factory Manager report also talked about having some way of viewing high scores, and that is something that would be supported by the database storing the scores associated with each username.

3 Ideas for Solutions

Should this project be redone for any reason, it would be best to use some established front-end framework like React or Svelte instead of vanilla javascript. It would save much time and effort to do so, and would allow the developers to focus more on making the actual game rather than simply getting a single page application to work.

Our planned way of implementing the database was via some form of relational database like SQL, which was to be hosted on a heroku server and interfaced with using node.js. The user would be able to serialize their gameState, and have it saved in the database, with the primary key being their username, thus allowing loading and saving existing games. The database would also be used for logging in.

4 Project Retrospective

One of the suggestions that could improve the rate of the product’s implementation is having multiple meetings per week. Initially, we started with meeting only once a week. After some struggles, meeting three times a week worked far better. However, even with them, we would recommend meeting even more in terms of frequency. This could mean four times a week or daily meetings. Of course, this would vary from group to group.

One issue we had was that since we were doing the project in an agile style, in the first scenario/sprints we just jumped into coding immediately with not so much long term planning. This led to issues where we had to refactor code and completely reimplement things that we initially created because as we developed more of the project, we found that our initial solution did not work. Having some more initial long term planning meetings would reduce this, because we would put off key features for later sprints and releases, and then when time came for those features to be implemented in a current sprint, we had to go through and plan and discuss it for the first time. Since we also didn’t have a long term plan around each feature, we often

had to cut down on features and simply work with what we had developed up to that point and the constraints that entailed.

Another possible way to improve the coding project process would be to have more in person coding meetings/pair programming. What would most often happen was a single person would work on their portion of the code, and then either they would bring it up in the next meeting, or they would ask some questions in discord. This led a bit to a fragmentation of effort where people would often have to wait long times to get responses on discord, or have little to show in meetings because they encountered an issue they needed help with. Getting together and coding in a group more often would have allowed us to more consistently get progress done, and possibly get through issues much more quickly. Even doing something like pair programming would have been a huge improvement, because not only does it lead to higher quality code and immediate resolution of many issues/bugs, it also means that at least two people, rather than one are getting an understanding of that part of the project. A big issue we had was that each person only really knew the part of the project they were working on, so when time came to connect all the systems together, it was a steep learning curve for everyone involved. Regularly working together on code would help to alleviate this.

The project prototype was created using vanilla Javascript, HTML, and CSS, which led to a lot of headache and difficulties in combining the logic and UI of the application. Since it was a single page application, our first implementation using multiple HTML pages would not work (without using something like session storage or cookies, which would be its own source of issues). We next tried using a map of pages to refer back and forth to, but found that this would cause the user interface to become out of date with the logic behind the application (i.e. the machine page would not show appropriately that some amount of products had been created or machines built). Our final implementation had us recreate the entire page based on the gameState, but this was a long and arduous process, with a more established front end framework it would have been much more streamlined. Alongside ease of use, there is some functionality in frameworks like React and Svelte that are not natively supported in Javascript, like Reactivity/Two-Way data binding, so we had to implement that from scratch (which had a much worse interface).

VII Glossary

Deliver: *A button for users to deliver resources and products they select in the Manger Delivery page.*

Do Tutorial: *A checkbox in the welcome page for users to turn on the tutorial mode so that when the game begins, there are messages to guide the users step by step to play with the game.*

Factories: *Factories contain slots for machines that the user can build. Production of products from resources via machines is calculated on a per factory basis, and they consume resources from the region-wide pool, and add products to the region-wide pool of products.*

Go Back: *A go back button for the users to go back to the previous page.*

Machines: *Used for making products. Different types can be researched, with different output rates and costs.*

Manage Deliveries: *A button where users can sell the Products they have and buy Resources needed to build Products.*

Map: *There are 3 regions in the map where users can choose which region they want to go and play the game.*

Region: *There are three different regions that the user is able to choose to play in. Each region has 3 slots for Factories, and Products/Resources/Trucks are shared across a given region, but not with other regions.*

Research: *Using money and time to research new products/machines/bonuses. Research is shared across all factories/regions.*

Resource: *An area in the Manger Delivery page in which users are able to select the amount and type of resource they want to buy.*

See Deliveries: *A button that opens up a pop-up window in which the users are able to see how much time is left in each truck's delivery.*

Stores: *An area in the Manger Delivery page in which users are able to select the amount and type of products they want to sell.*

Supplies: *A button that causes a pop-up window to appear showing everything the users have including the amount of product, resource, factory status and money.*

VIII References / Bibliography

[1] Amazon, "Amazon Web Services," [Online]. Available: https://aws.amazon.com/. [Accessed

2019].

[2] Draw IO, "draw.io," [Online]. Available: https://www.draw.io/. [Accessed 2019].

[3] Robertson and Robertson, Mastering the Requirements Process.

[4] A. Agrawal, K. Ahuja, R. Miramontes and Z. Sajith, “Factory Manager Project Report,” Group 2 - Factory Manager - Final Report.pdf, 2019.

[5] A. Silberschatz, P. B. Galvin and G. Gagne, Operating System Concepts, Ninth ed., Wiley, 2013.

[6] J. Bell, "CS 440 Fall 2019," 8 2019. [Online]. Available: https://www.cs.uic.edu/~i440/.

[Accessed 2019].

[7] J. Bell, "Underwater Archaeological Survey Report Template: A Sample Document for Generating Consistent Professional Reports," Underwater Archaeological Society of Chicago, Chicago, 2012.

[8] M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

[9] D. Zhang, T. Wang, L. Repta and M. Apreza, “Factory Manager Full Game Scenario,” group28FullGameScenario.pdf, 2022.

[10] D. Zhang, T. Wang, L. Repta and M. Apreza, “Factory Manager Tutorial Scenario,” Group28FactoryManagerScenario.pdf, 2022.

IX Index