Space Economy Simulator Manual

Cal Poly Computer Engineering Senior Project 2018

Joseph ALFREDO
Kevin YANG

Abstract

This project is a space simulation game focused on trading and the economy of different planets. The player takes control of an interplanetary corporation, headquartered on space station, aiming to score as much profit as possible while maintaining the station. Player actions include buying and selling goods from a trade route between planets, upgrading the space station, traveling to different trade routes, and deciding response to key events. The game ends when the player runs out of money. The final score is then calculated, based on how far they are progressed in the game.

Introduction/Motivation

The motivation of this project was that creating a game from scratch would be a good culmination of the knowledge and skills of software gained at Cal Poly. This would be a good application for object oriented programming along with scripting to bring them together. Additionally, designing and planning a game would be a good way to demonstrate our organization abilities. Finally, since both of us has limited experience in creating games, this project would be a good learning opportunity. This project aims to educate players about basics of international trade, supply and demand, and how a player acting as an international corporation needs to work together with other planets to ensure a stable economy for nearby planets. This game will simulate basic supply and demand, by including price fluctuation, inflation, deflation and economic sanctions. Richer planets will often times take advantage of poorer planets, due to the existing mechanics of trade. However, if both planets were to not be in poverty, the overall wealth of the systems would be much greater. This is similar to the case of prisoner's dilemma, where each planet has incentive to profit at the cost of others, but cooperation will do the greatest good for everyone together.

Implementation

Design and Production

This project introduces several challenges for design implementation. First, there needs to be a model for a dynamic economy system, which includes price fluctuations, different supply and demands, and goods stockpiling. Second, the user interface that enable clear representation, interaction and analysis of a dynamic economic system. Third, a dynamic planet generation which ensures replayability with a dynamic simulation and gameplay experience. Finally, an enjoyable gameplay experience, with a goal, different interesting options, challenge and uncertainty in some outcomes.

This project uses the Unity game engine as the framework of our game. As stated before, we had a little experience with this engine, but most of the times we were learning as we went. This engine was chosen because we were familiar with Java and object oriented programming which is similar to C#. Unity was also very intuitive to use and learn. There was also the option of finding additional assets to use in the Unity store to do some of the work we could not, such as game music and sprites.

In order to collaborate and work on the game together, we mostly used Unity Collaborate and some Git. Unity Collaborate was mostly used because of its simplicity and built-in functionality with the Unity Engine. However, since it was so simple, it lacked many of the features we were familiar with Git, such as merging files, displaying changes, and multiple branches. For that reason, Git was also used to store our scripts as a backup and for help in merging future changes.

Code Organization and Main Game Loop

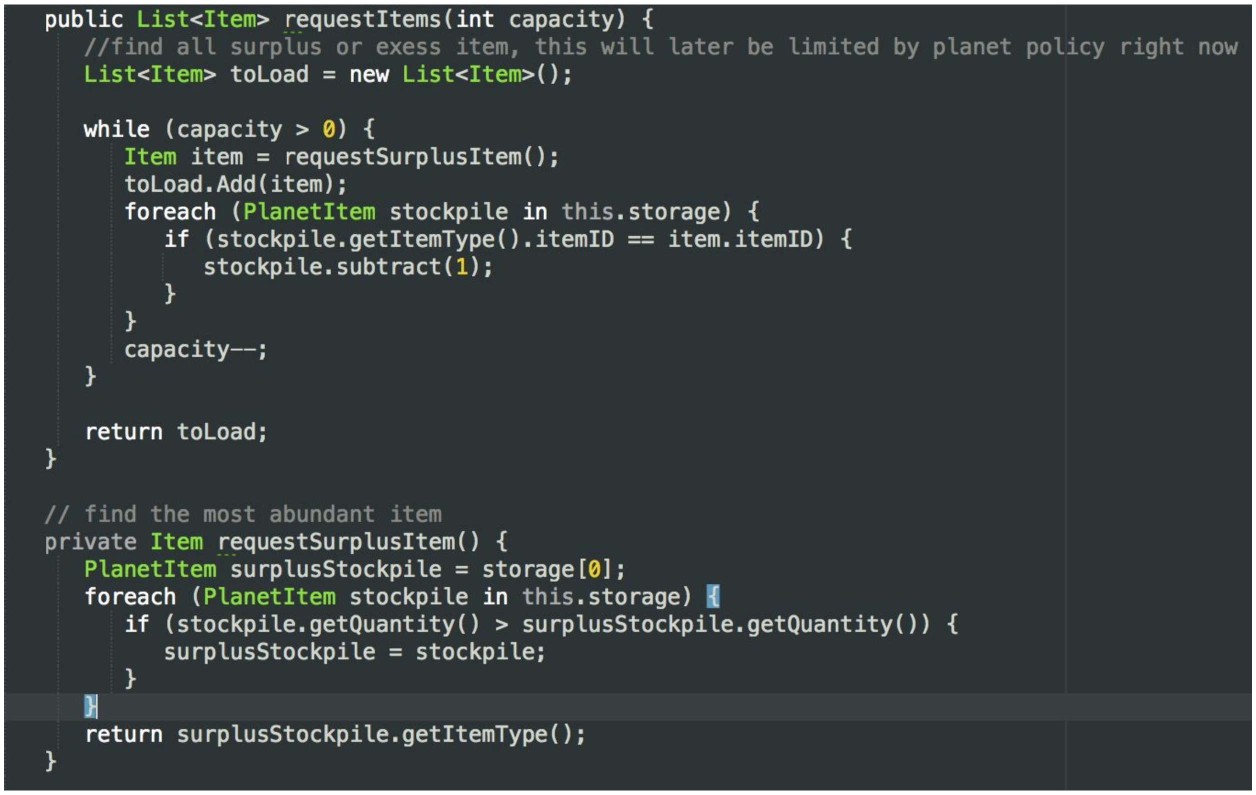
The main script that pieces together all the others is the game handler script. It has access to all the generators, such as the planet and ship generators, that it could tell at the correct times to generate more instances of a certain game object. There are also many event listeners, such as on click of a docked ship, that the game handler must subscribe and handle. The main game loop starts with the game handler telling the generators to generate the starting map and then subscribing to all the events needed. Then every day is started by ships being sent out by every planet while the game handler counts down the time. Once the time has ended, the game handler would then switch into the end of day report screen to allow for player actions before the next day. This is then repeated until the game is over when the player has run out of money. One possible upgrade to this design would be to implement the game handler as a finite state machine, making the defined states and code for each state much more clear and precise.

Economy System and Planets

The driving force behind our economy system based on the each planet having its own local stockpile of goods which will affect how much each type of good is worth for that planet. This will ensure dynamic pricing for the same type of goods across different planets and different times. Planets start with a different amount and type of goods produced and consumed, reflecting the asymmetrical nature of international trade. Every day, all planets will produce and consume a specific amount of goods. Prices for each good will be calculated based on the scarcity of that good on the planet compared to the entire planet stockpile relative to the equilibrium or 50% of the stockpile. The function for price is the following:

$$Final\ Price = |Current\ Stock - Max\ Stockpile/2| \times Base\ Price\ +\ Event\ Modifier$$

And here’s the code

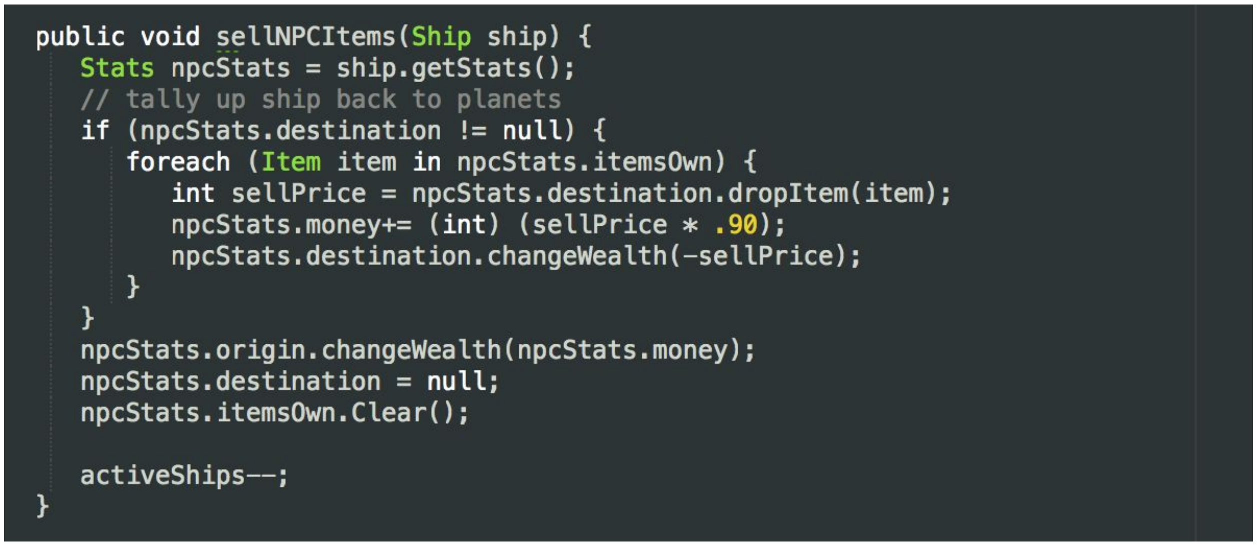


```
public List<Item> requestItems(int capacity) {
    //find all surplus or exess item, this will later be limited by planet policy right now
    List<Item> toLoad = new List<Item>();

    while (capacity > 0) {
        Item item = requestSurplusItem();
        toLoad.Add(item);
        foreach (PlanetItem stockpile in this.storage) {
            if (stockpile.getItemType().itemID == item.itemID) {
                stockpile.subtract(1);
            }
        }
        capacity--;
    }

    return toLoad;
}

// find the most abundant item
private Item requestSurplusItem() {
    PlanetItem surplusStockpile = storage[0];
    foreach (PlanetItem stockpile in this.storage) {
        if (stockpile.getQuantity() > surplusStockpile.getQuantity()) {
            surplusStockpile = stockpile;
        }
    }
    return surplusStockpile.getItemType();
}
```

All planets will prioritize buying goods they consume and selling goods that they produce. Each day, trading ships will be sent to surrounding planets in an attempt to fulfill their planet demands and sell their own cargo. Each ship will be filled with excess goods, which are above the 50% of stockpile equilibrium. This greedy algorithm will find all surplus items to export to nearby planets, which is also affected by a planet’s policy from events. This simulates the international politics affecting international trade. The algorithm will start with finding the most abundant item and pushing them into the ship storage. Each planet then will buy all of the items arrived via ship the arrived to their planet. The planet will then pay for each good using their local price, while the planet the ship is from will take a small tax or tariff. The function for profit is the following:

$$Final\ Profit = FinalPrice\ *\ (1\ -\ \frac{Tariff\ Percentage}{100})$$

And here's the code



```
public void sellNPCItems(Ship ship) {
    Stats npcStats = ship.getStats();
    // tally up ship back to planets
    if (npcStats.destination != null) {
        foreach (Item item in npcStats.itemsOwn) {
            int sellPrice = npcStats.destination.dropItem(item);
            npcStats.money+= (int) (sellPrice * .90);
            npcStats.destination.changeWealth(-sellPrice);
        }
    }
    npcStats.origin.changeWealth(npcStats.money);
    npcStats.destination = null;
    npcStats.itemsOwn.Clear();

    activeShips--;
}
```

Another attribute of planets are their wealth, this wealth is what puts them in 3 tiers: Poor, Developing, and Prospering. The higher a planet's tier, the more goods their ships can carry. With more goods on ships, means more profit per ship sent out by this planet. These tiers also influence the amount of goods produced each day on the planet. Higher tier planets have a higher rate of production to support their increased capacities on the ships.

User Interface

Each planet has a pop up graph that hovers near it. This graph includes all the information about recent prices of goods and wealth of that planet. The player could toggle between displaying either wealth or any of the items by pressing the toggle button on the upper-right hand corner. When switched to displaying items, the item displayed can be changed by pressing the item icons on the right. All of these graphs display the prices in the last 5 days and is constantly updated at the end of each day.

To ensure good experiences, the player is able to open a trade screen when they click on the ships that are docked, at the bottom right of the screen. This will pop out a trade screen that allows the player to drag and drop items from the player inventory to the opposing inventory to determine which items they want to sell or buy. After the player is done the transaction money will be subtracted or added accordingly.

At the the end of the day, the player is able to see all of his/her statuses listed on the daily report screen. Here upgrades are listed on the center of the screen, this list is dynamically generated to apply modifier to various player status. Another option is to go to travel mode and choose a different travel spot available to the player, if the prerequisites are met, like having enough money to perform the jump.The starting location is randomized at the start, so a player needs to strategize their resource usage to obtain higher scores.

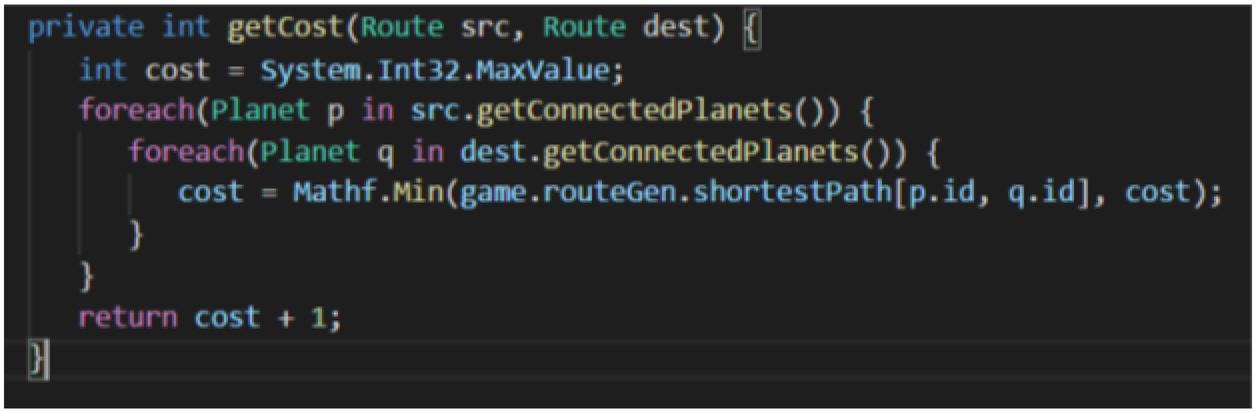
Map generation

Possible planet locations and routes are loaded in through two separate .csv files. This file is then by parsed for x and y coordinates assigned to every planet location and assigning each an id. With these locations, the planet generator will generate every planet to have a random name, starting wealth, and list of produced and consumed items. This planet generator will now store all these planets in a list. To parse the routes into the game, each row of the .csv file has a number of planet ids and all of those corresponding planets would form the route. The route generator is responsible for generating these routes after parsing the file and would also store a copy of all the routes in a list. To model a route, the object we created would just contain a list of connected planets. With these connected planets, the midpoint between all these planets could be calculated, stored, and used as the position a player's space station would land if traveled to. The player's starting route is randomized as well. The purpose of this is to provide replayability and a more dynamic feel to the game.

Travel Mode

Traveling to different routes is an important part of the game. Different routes not only have different planet's connected to them, but also a different amount of planets connected. The algorithm used to implement the travel cost was Dijkstra's. Every hop would be a cost of 1 and that cost would then be multiplied by the player's jump efficiency that could be improved in the upgrades or also affected by events. The results of all these hops from all the planets to all other planets were stored in a 2d array for easy access and wouldn't have to be recalculated each time.

One challenge to using Dijkstra's was that it could only be properly implemented based on the adjacency matrix of planets not the routes themselves. It took some time to reason it out, but in the end it was simply the following:



```
private int getCost(Route src, Route dest) {
    int cost = System.Int32.MaxValue;
    foreach(Planet p in src.getConnectedPlanets()) {
        foreach(Planet q in dest.getConnectedPlanets()) {
            cost = Mathf.Min(game.routeGen.shortestPath[p.id, q.id], cost);
        }
    }
    return cost + 1;
}
```

Results

Game Play Experience

The player can influence the performance of planets by buying or selling goods. The player's goal is to obtain as much money as possible before reaching bankruptcy. The player's final score will be calculated based on maximum amount of money multiplied by how long a game last. This will motivate player to take risks, as it is most efficient way to gain money, with the possibility of ending the game prematurely.

There are also multiple events which require careful management of their resources. This includes, but not limited to, fuel to jump around trade routes, planet price changes, and increased station maintenance. Finally, each in game day cycle will provide the option player to move around the galaxy and purchase upgrades to further assist the goal of maximising score. These upgrades will add value, which make the challenges easier to solve and give player the sense of progression and accomplishment. To ensure scalability, player status is implemented as a dictionary with enum of status as the key. To apply effect to these status using the enum as the key particular effect could be applied to player status. Event can be setup as modifier to these status, lie fuel and happiness. Some status like happiness works in the background which affect the probability of good or bad event get tby the player, this ensure player control on the random element of the events system.

Performance/Improvements

Overall, the project finished a large portion of the milestones laid out, including several major functions that were not planned for in the initial sprint plannings, such as larger galaxies and the random generation of them. Some of the milestones that had to be cut out, either due to time or an obsolete design, were multiple tiers for goods, combination of goods, and post-game unlockables. The multiple tiers and combination of goods would have been a large upgrade to our game, but it would also be a large commitment of time to implement. It could have added another layer of complexity and interaction by having certain planets produce upgraded goods, but only with the resources of other planets.

There was a major pivot from a more roguelike aspect of gameplay, with focuses on difficulty and unlocking more and more content throughout game plays. We changed to a more simulation focused game, due to the unexpected complexity of creating a simulation game with all its interacting and moving parts.

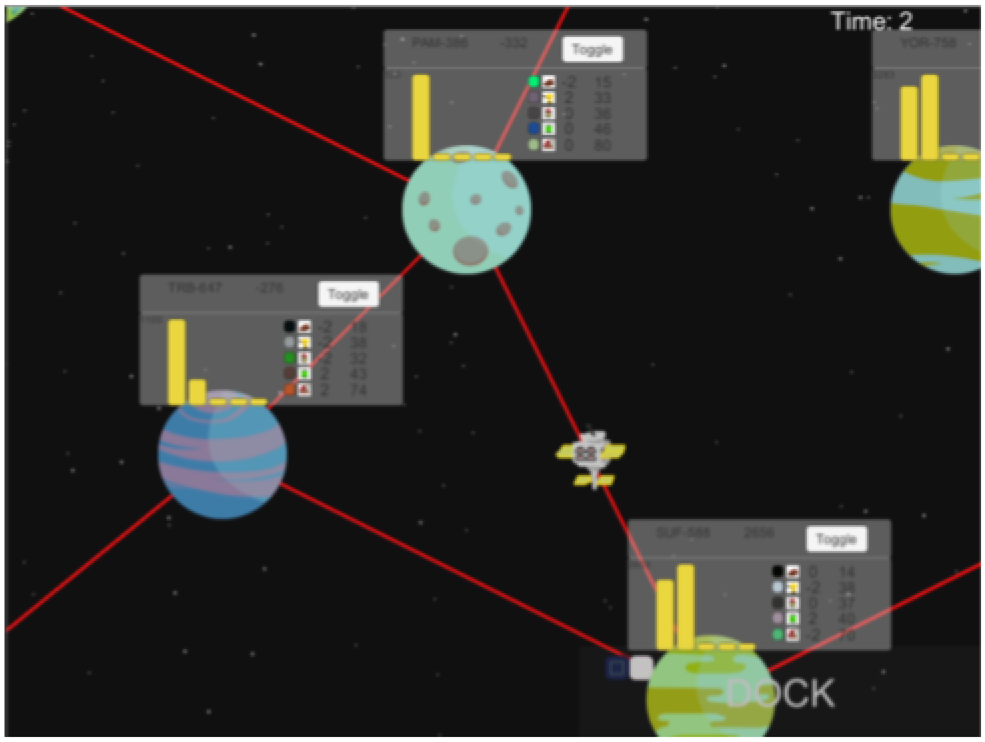
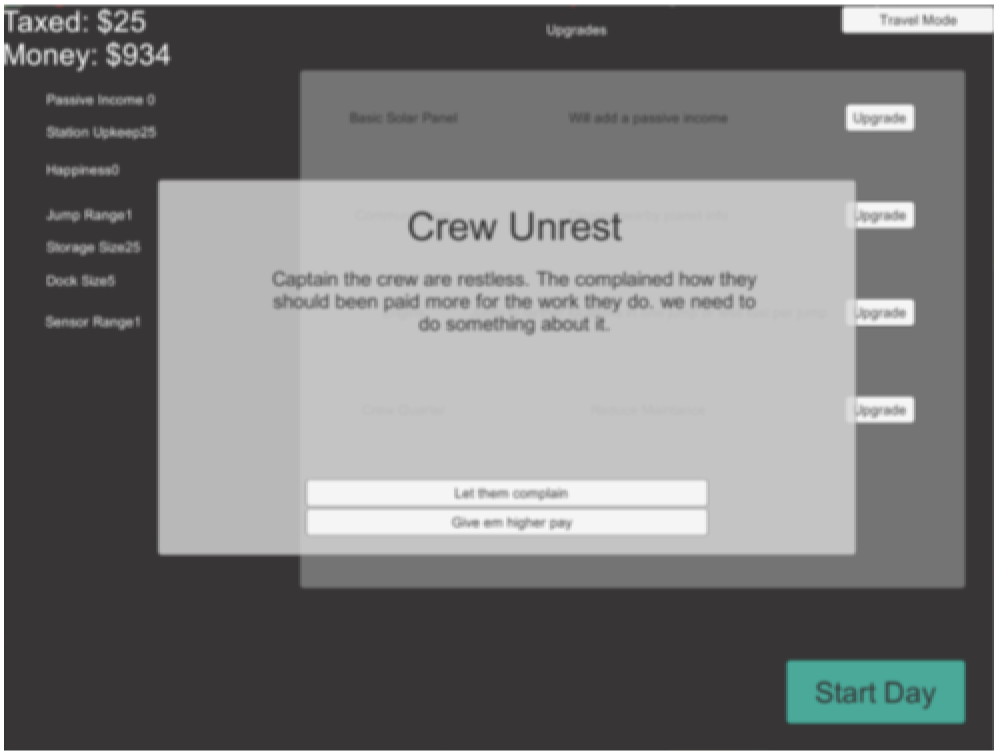


Figure 1: Player's view of the galaxy



Crew Unrest

Captain the crew are restless. The complained how they should been paid more for the work they do. we need to do something about it.

Figure 2: Example event that occur at the end of the day

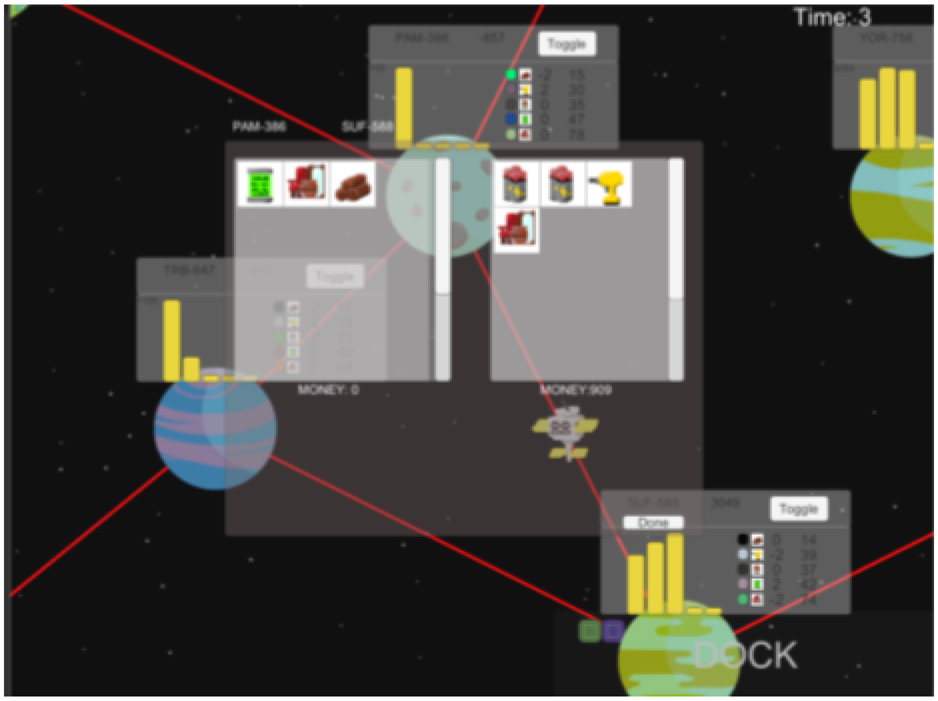


Figure 3: Trade screen