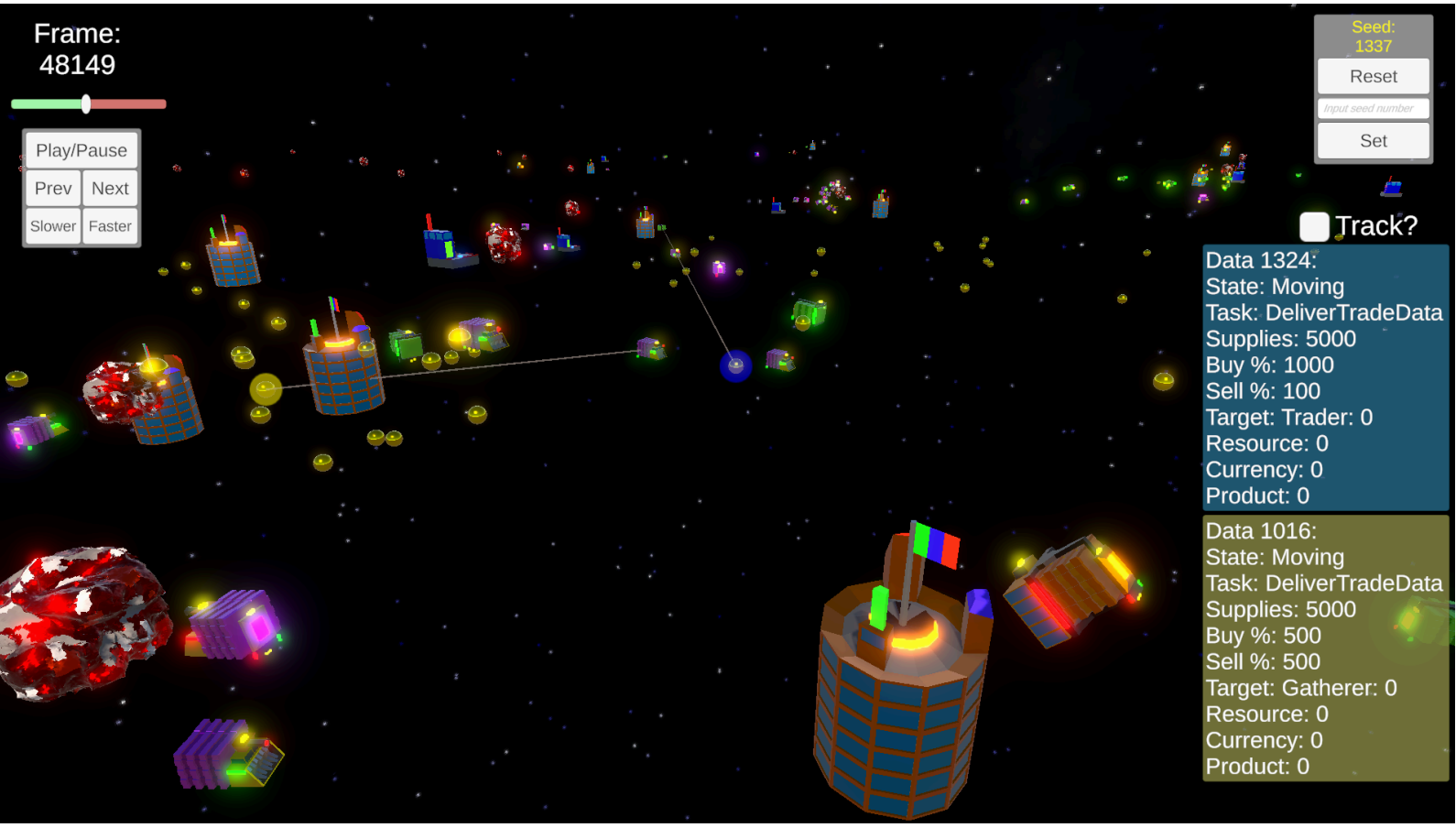
Deterministic Multi-Agent Simulation



Jørgen Tandberg

Yrgo, Game Creator 2024

Gothenburg

December 2025

Preface:

I would like to thank my family for their patience, my friends for dealing with my rants and Matte for the encouragement!

Table of Contents:

Preface:..... 1
Table of Contents:..... 1
Introduction:..... 2
Implementation:..... 3
    Development:..... 3
    Lessons and technical challenges:..... 5
Result:..... 6
    The Simulation Explained:..... 6
    In conclusion:..... 8
    Future Plans:..... 9
Bibliography:..... 10

Introduction:

Purpose:

The purpose of this project is to design and implement a deterministic multi-agent simulation system that produces complex interactions without relying on randomness. The project also aims to explore how playback tools such as rewind, fast-forward, time-skipping, and save/load can be integrated into a deterministic environment to enable accurate reproduction of simulation states across different sessions.

Background:

I once made a simple traffic system in Unreal and it was very fun to get it working. I've also been wanting to simulate a giant galactic economy for future space games I plan to make. I've drawn a lot of inspiration from the Youtube channel Primer and ant simulation videos like this one. The idea for large scale simulation has been something I've been thinking about since I first learned programming. The idea to make it deterministic came after listening to a podcast. In the podcast the developer talked about how he had built a custom deterministic physics system in Unity, including how this automatically gave him the framework for a replay system, bug replication and many other perks.

Questions:

How can a deterministic simulation system with multi-agent interactions be built to support fully reproducible playback features such as rewind, fast-forward, and time-skipping?

Limitations:

The simulation will focus on simplified agent behavior and interactions, not on full economic or galactic-scale depth. Visual presentation will be minimal; the core priority is system behavior and determinism. Networking, parallel processing, and distributed simulation will not be implemented.

Implementation:

Development:

The problem with the Update() method:

The Update() method and Time.deltaTime relies on drawn frames and frame times, this makes it hard to achieve deterministic results with it. Even if you cap the framerate to a number your system always can handle, each drawn frame will take a different amount of milliseconds to draw. If the system was based on Time.deltaTime it would never be able to go to the exact same point in the simulation. If the system was based on drawn frames the entire solution would freeze. These issues might be solved with a work around, but the system would still be tied to factors out of the simulation's control.

Note: When I'm using the word "frame" in this report I'm not referring to the drawn frames of the engine. I'm referring to each step simulated by the simulation system.

The way to determinism:

Handler.cs is intended to replace Unity's built-in Update() method.
Object.cs is the class that represents the agents, it's attached to every game object you see in the scene.

Handler.cs contains the variables current frame, target frame and a pool of all agents. When the target frame is increased by 1 it will loop through each agent, calling the method NextFrame() on it. Object.cs contains the agent's current state, stats and configuration. Based on its state and configuration the agent calls the corresponding Behaviour.
Once all agents have been looped through Handler.cs will update its current frame by +1.

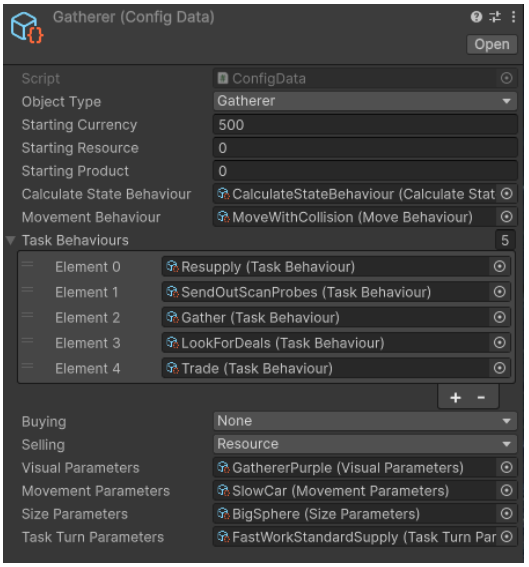
```
private void SendFrame()
{
    List<Object> currentObjects = new(activeObjects);
    foreach (Object obj in currentObjects)
    {
        obj.GoToFrame(currentFrame);
    }
}
```

```
public void GoToFrame(int frame)
{
    if (frame == objectStats.localFrame)
    {
        return;
    }

    Behaviours();

    RefreshLineRenderers();
}
```

To enable fast creation and spawning of varied initial conditions. So I added a seed system to the Handler.cs. To ensure that the seed didn't get modified by Time.deltaTime the seed state is saved after spawning and loaded before spawning. This makes it possible to create several simulation results while also keeping the results deterministic.

Agent Behaviours:

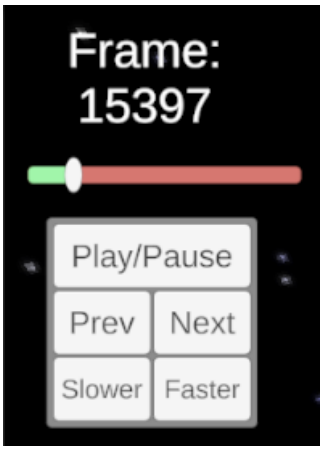
To make agent behaviours easy to add and remove from any one of the agent types. Each behaviour takes in the calling agents stats, performs the behaviour and returns the updated stats. The movement behaviour for example will read the agent's position, movement speed, collision parameters and target from the stats, it uses these to calculate the movement direction, checks for collisions and finally applies the movement. It then returns the modified stats back to the agent so it can store them.



The behaviours are sorted in lists for each agent state. When the behavior is being performed the logic will also check if it should continue the current behavior, in the case of movement, if the target is within range it will let the agent know that the current behavior is done and the agent will move on to the next behaviour in its configuration.
Many types share behaviours, like movement, trade, scan etc..

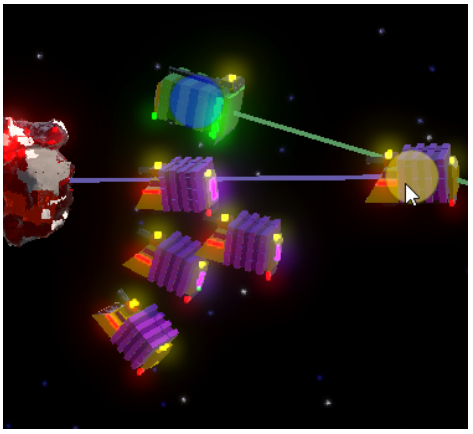
Playback:

There is a slider that shows the timeline and it can be clicked to skip to a point in time. When skipping forward more than one frame the handler will simply repeat the process until its current frame equals the target frame. If the user decides to go backward one frame or more, the system will re-initialize the simulation and simulate forward from frame 0 until the current frame equals the target frame again. There's a play button that starts a coroutine which will repeatedly add +1 to the target frame at intervals the length of the playback speed. The playback speed can be modified by clicking the faster/slower buttons on the interface. To prevent the UI or user actions from altering the simulation an event was set up in the Handler.cs that the UI can listen to.




```
void SimulateFrames()
{
    if (targetFrame < currentFrame)
    {
        ReInitializeSim();
        return;
    }

    for (int i = currentFrame; i < targetFrame; i++)
    {
        ChangeFrame(1);
    }

    Actions.NewFrame.Invoke(currentFrame);
}
```

Collision and avoiding randomness:



To handle collisions a method was added to Handler.cs that allows any agent to check if there is another agent in the position specified, they would feed in the position they are trying to move to and the method will tell them if the area is occupied or available. The collision system uses the radius of the agents to determine the distance needed between any agents at the targeted position.

If the position is occupied, the agent needs to set a new direction without randomization. This was solved by adding an int parameter. It would toggle between a 1 and -1 whenever a move attempt failed, storing the state for the next frame, causing the agent to move left or right depending on the negative or positive value.
This worked, but it was later discovered that it could cause agents to endlessly shuffle left and right. It was then improved upon by counting up every failed movement attempt, using modulo to try alternating tactics; moving left or right and up or down at increasingly steeper angles, if nothing works the agent will wait until the next turn and try again.

Lessons and technical challenges:

Selecting a scenario:

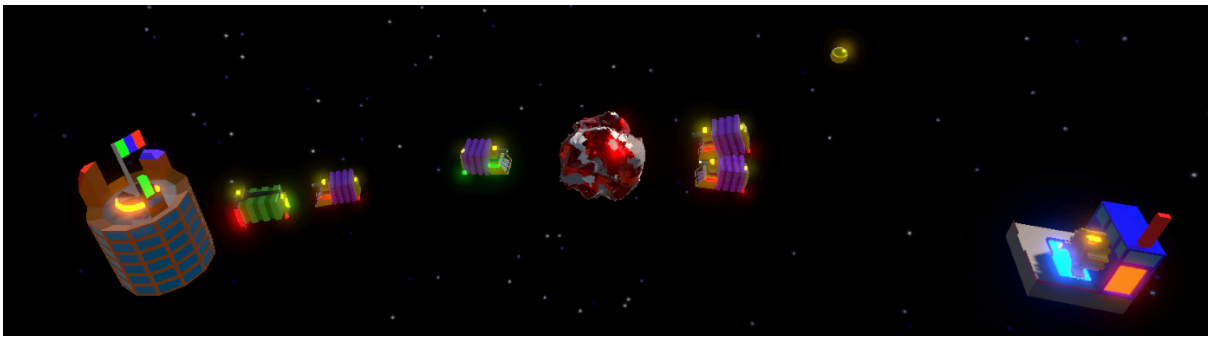
The plan was to design a tool that could simulate any scenario. Scenarios like cities, traffic, forests and more were considered. To streamline the project the choice was made to simulate a simple galactic trade economy with resource gathering and production. A straightforward scenario which could easily be expanded upon.

Playback performance and Save/Load

While fast forwarding or skipping time or going frame by frame does work, the performance is terrible when skipping large sections of time. The hope was that with implementation of DOTS the performance could be improved hundredfold. This would probably be the best option if given enough time to develop the system. There were also plans to improve rewinding and skipping to frames already simulated by adding save states at set intervals. These two solutions combined would surely improve the performance of the simulation. However they were not strictly necessary and were deprioritized.

Result:

The Simulation Explained:

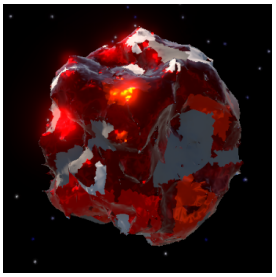


The Handler:

Is the core of the simulation. It spawns and initializes all agents and pools them into the necessary pools. It keeps track of the current frame and target frame and cycles through each active agent on the next frame. It contains methods to assist agents with checking collisions, spawning, market prices and much more.

The Objects:

There are only three classes that exist in the world of the simulation, they all inherit from a base class called Object.cs. The classes contain logic that can have effects on other agents in the simulation. They do this by moving, performing tasks and exchanging information. The different Object classes all have different overrides and custom methods. These are the three Object classes and their types:
FixedObject: *Resource Spot, Refiner, Station.*
MoveableObject: *Settler, Gatherer, Trader*
DataObject: *Data*

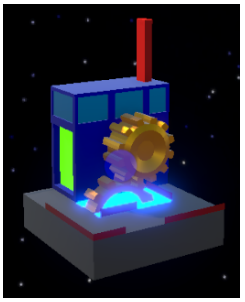


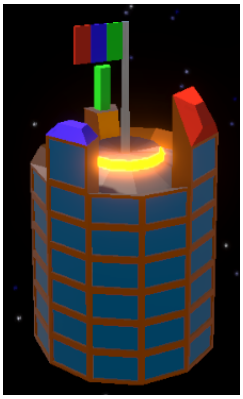
The Resource Spot:

These are spawned during initialization, at locations determined by the seed. They contain resources which can be gathered and will despawn when depleted. These are the targets for expansion and as such Stations and Refiners will be built in its proximity.

The Refiner:

Buys resources and produces products with it. It then sells the product for a profit.



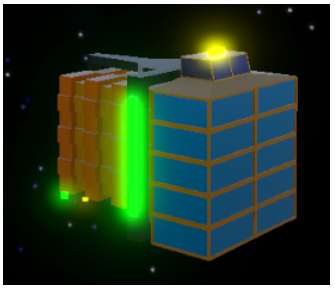


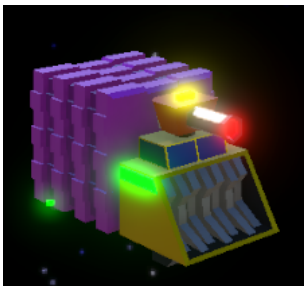
The Station:

Acts as a home base for ships. It buys products and sells them as supplies for ships docking there. It also uses products for producing new Traders, Gatherers and even Settlers. The choice of which agent it will produce is done by a static formula consisting of an int counter and modulo. Every 5th agent spawned will be a settler until it has spawned a maximum of 4. Every 9th is a trader, the rest are gatherers.

The Settler:

Aside from Resource Spots there is one Settler spawned at initialization. The settler will scan for the closest unclaimed Resource Spot, move there and instantly build a settlement containing a Station, Refiner, Trader and Gatherer. The Settler will then be destroyed.



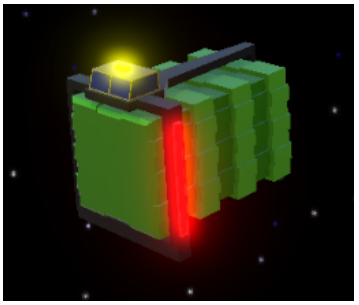


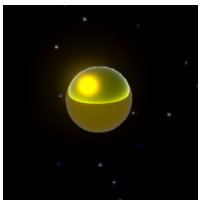
The Gatherer:

Scans for nearby Resource Spots and starts gathering resources there. Once its cargo is full it will look for the best trade deal and sell them. It has a station as its home point and it will go there to resupply as needed.

The Trader:

Waits for the cheapest and nearest trade deal offering products. Once it has a full cargo of products it will look for the most profitable and nearest buyer. It has a station as its home point and it will go there to resupply as needed.





The DataObject and Data:

The data agent was made to visualize the data exchange between agents. The intention is for it to represent radio signals moving at light speed through space. It helps increase the variation by making distance a factor in information exchange. It's lightweight and only active during the time it is used.

Data will be spawned by other agents with a target agent, it will bring either *Trade Deal Data* or *Scan Data*.

Trade Deal Data is sent from Stations and Refiners to possible buyers and sellers of goods they need. This is used by the Gatherers and Traders to locate the best place for them to buy and/or sell goods.

Scan Data is sent from Settlers and Gatherers to look for Resource Spots that are nearby. The Resource Spots will then bounce back the Scan Data adding information about itself to the originator. If no suitable target is found they will increase the scan range and try again.

Added Complexity:

Distance is the first and only true factor of variation in this simulation. The positions of the Resource Spots and the position of the initial Settler determine how everything will go. But there are two other factors that help increase the variation, Trade Prices and Collisions. If you look deeper at this functionality you will realize they are also the result of distances, but they do add more variation as they are a result of the effects distances have had earlier in the simulation.

Trade Prices: Stations and Refiners modify the percent profit/loss they are willing to accept depending on how much resources and/or products they have. When the Traders and Gatherers receive their updated prices they may change course in favor of a better deal.

Collisions: Without collisions every journey would be a straight line, the fact that other agents block the path adds a lot of variation in the long run.

Playback and UI:

The UI and camera navigation tools allow for great control and transparency of information for the user. It has an input field allowing the user to input new seed values to start the simulation from.

In conclusion:

The simulation is 100% deterministic and it can generate different variations based on a seed. When all the different agent types and interactions come together, the simulation is quite complex and can grow to a large scale. Time skipping and playback functionality is there. However saving/loading did not get implemented in time.

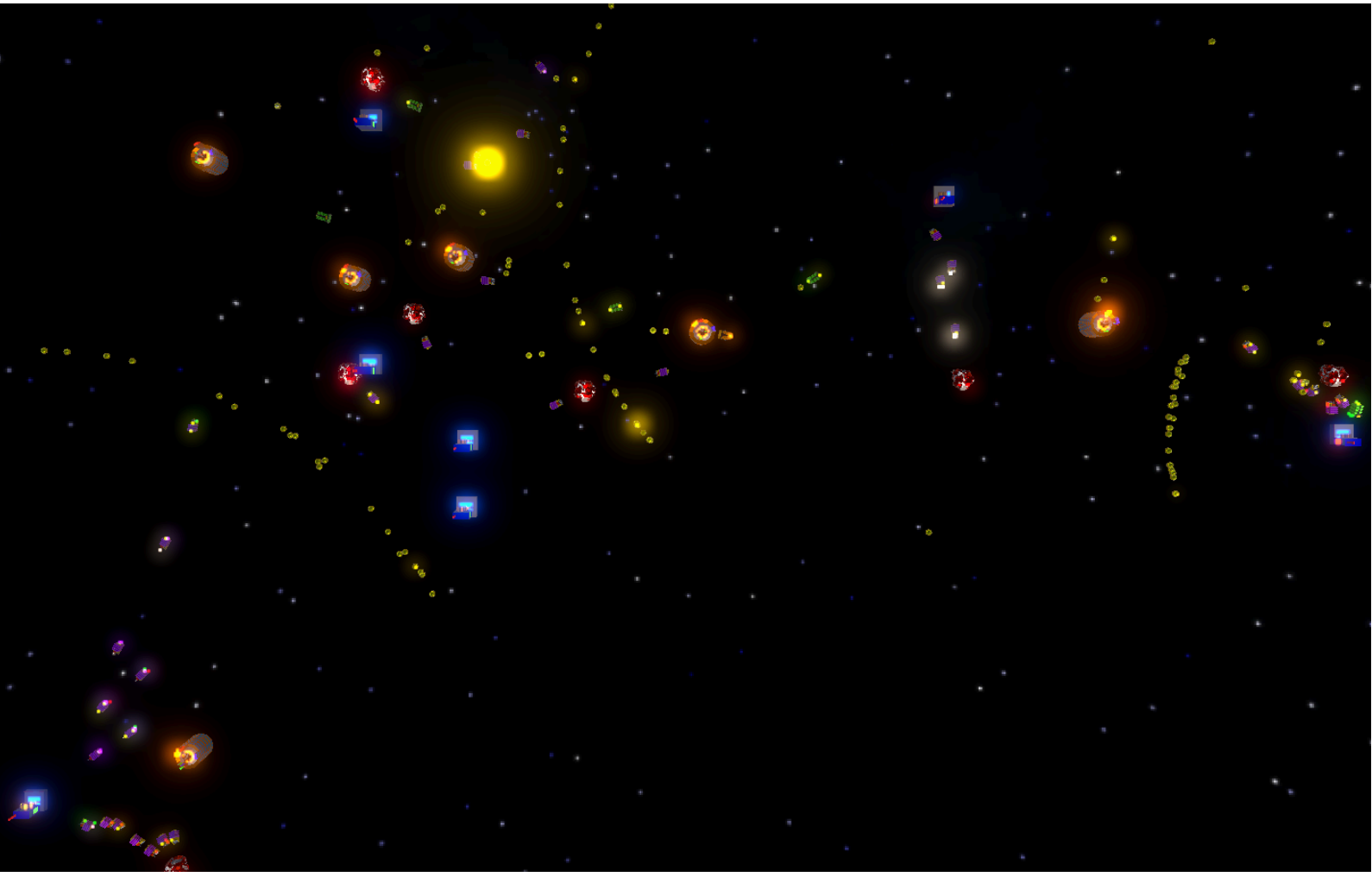
It was possible to make this solution by following a traditional Unity code structure, however its weakness is performance. Restructuring the same system for ECS/DOTS would most likely be a very good solution to the proposed problem.

Future Plans:

Improvements to the user interface; organizing it better and sorting information into different windows, making it easier for the user to get the information they want. Adding popups showing where trades or other events happen, including information about the event.

Reworking the simulation to use mostly ECS/DOTS and adding autosave at intervals to improve the performance.

Adding a more complex market and trade system that affects prices galaxy wide and locally. More resources and product types, creating a longer more complex supply chain.



Bibliography:

Primer youtube: https://www.youtube.com/@PrimerBlobs
Inspiration podcast: https://youtu.be/M-Ihm11KsjU?si=PHOhFiYd5CDqD5G-
Ant simulation example: https://www.youtube.com/watch?v=GSTkz_6UVds
Code monkey DOTS tutorial: https://youtu.be/4ZYn9sR3btg?si=dYNhBXHYvky7ivQ8
Unity manual: https://docs.unity3d.com/6000.2/Documentation/Manual/UnityManual.html