Integration of Python Combat Simulator into the Vietnam Battlefield Modelling Project

As part of this group project, I contributed extensively to the modelling and simulation of battlefield conditions using Mathematica. This included data processing and visualization of real casualty data, terrain-based impact modelling, and dynamic animations reflecting force depletion over time. However, I also expanded the scope of my contribution by developing a Python-based simulation environment to reflect resource management, troop decision-making, and live battle conditions in a more interactive and visual format.

The Python program is built using **Tkinter** for GUI elements, **Matplotlib** and **Pygame** for visual battle simulations, and **JSON** for persistent user data tracking. The interface allows a user (playing as a military commander) to allocate resources across battlefronts, adjust deployments, and view attrition reports and logistical demands. A built-in AI system determines how the opposing side (PAVN forces) responds, using probabilistic thresholds to decide whether to attack, defend, or reinforce. This AI decision-making is based on current troop disparities and acts as a model of responsive adversarial behaviour.

In the second half of the code, battle simulations are visualized in 3D using Matplotlib and in 2D using Pygame. These simulations model reinforcement rates, attrition, and proximity-triggered engagements. While simplified, they mirror the mathematical structure explored in Mathematica — such as exponential force depletion:

$$\frac{dF}{dt} = -kF(t)$$

and growth-corrected reinforcement:

$$N(t) = N_0 e^{rt}$$

The game logic also incorporates combat efficiency estimations and stochastic losses based on intensity randomisation. Additionally, the simulator introduces dynamic logistics calculations such as food consumption, fuel requirements, and weather-based supply delays. These logistical variables reflect real-world military considerations drawn from historical air-drop and supply chain data discussed in supporting literature (e.g., Gruenwald 2016, Rich 2015).

What makes this addition valuable is not just its interactivity, but the way it complements the existing Mathematica model. While the Mathematica side of the project focused on data visualization, aggregation, and terrain-linked casualty analysis, the Python module allows that data to be placed in a **game-theoretic and resource-aware context**. This bridges the gap between static analysis and user-driven simulation.

Limitations and Future Expansion

This simulator is, by nature, a prototype. Full scientific or operational-grade models require years of iterative development, complex tuning, and validation — I had only four months, while simultaneously working on the analytical side in Mathematica. As such, there are limitations:

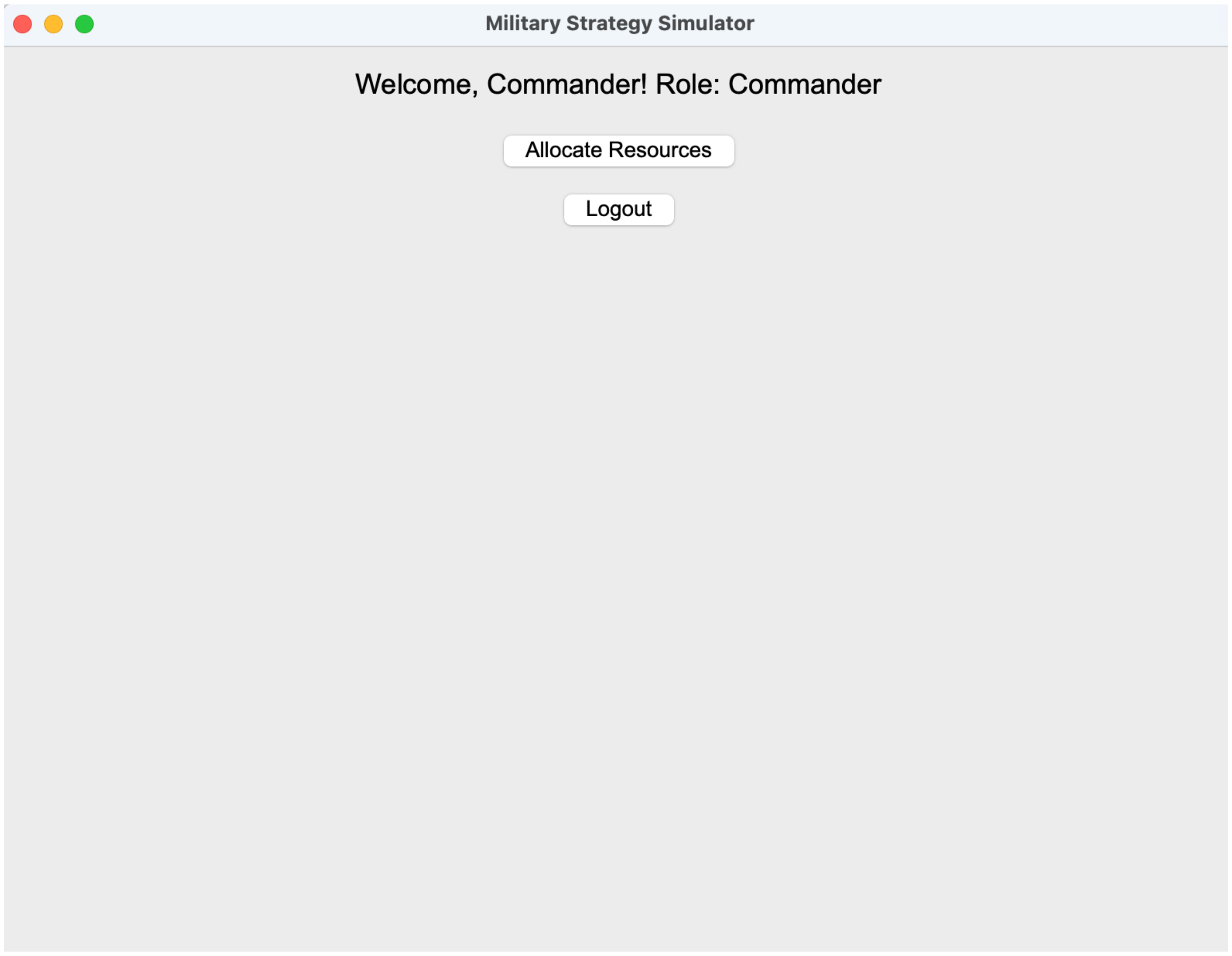
- It does not yet integrate real-time data from the CSV files directly.
- Battlefronts are abstracted into a simplified three-sector layout.
- No map-based terrain modeling is yet implemented in the GUI.

However, the framework is **modular and extendable**. In future iterations, the simulator could ingest real terrain data from the terrain.csv and weather.csv files, directly reflect historical battle logs from the Vietnam dataset, and model logistic depletion based on actual recorded conditions. It could also be adapted for reinforcement learning models or AI training scenarios.

Summary

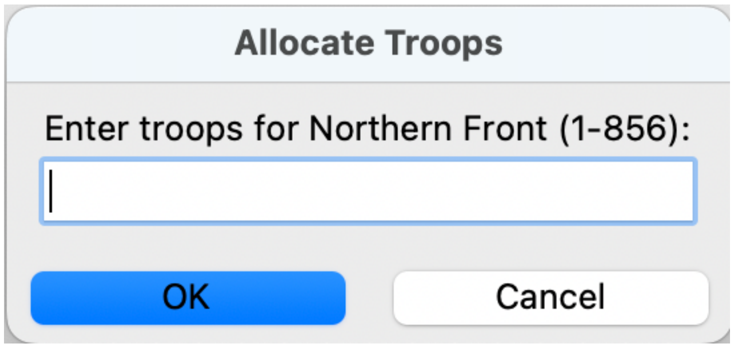
In conclusion, this Python tool enhances the project's modelling dimension by giving it a strategic interface where mathematical concepts meet decision-making. It reflects real constraints (supply chains, weather, attrition) and introduces an adversarial component (AI) into the battlefield model. Most importantly, it offers a visual, interactive form of engagement that complements the structured, static results from Mathematica. Together, the two systems form a bridge between historical analysis and applied simulation.

Below are the testing screenshots and a justification of each step:

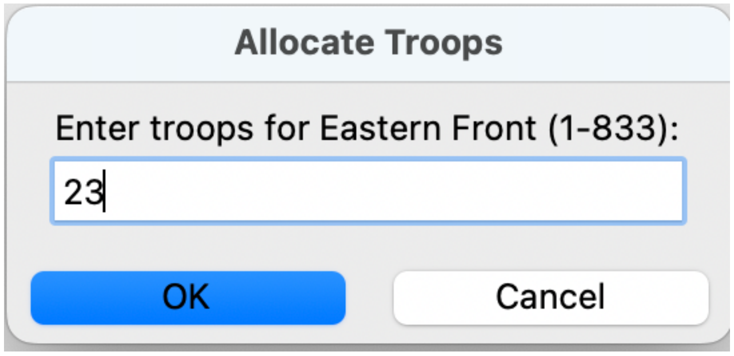


Welcome, Commander! Role: Commander

This is the first page that opens, I click the first button:

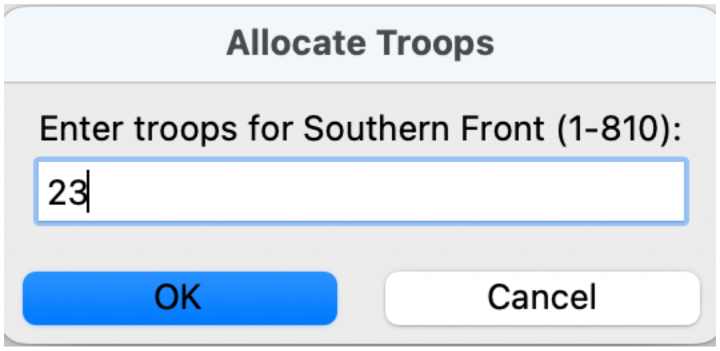

Allocate Troops

Enter troops for Northern Front (1-856):


Allocate Troops

Enter troops for Eastern Front (1-833):

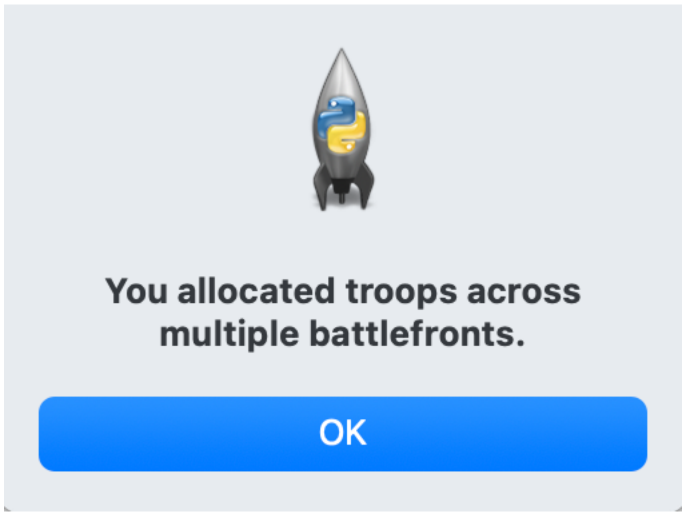
23


Allocate Troops

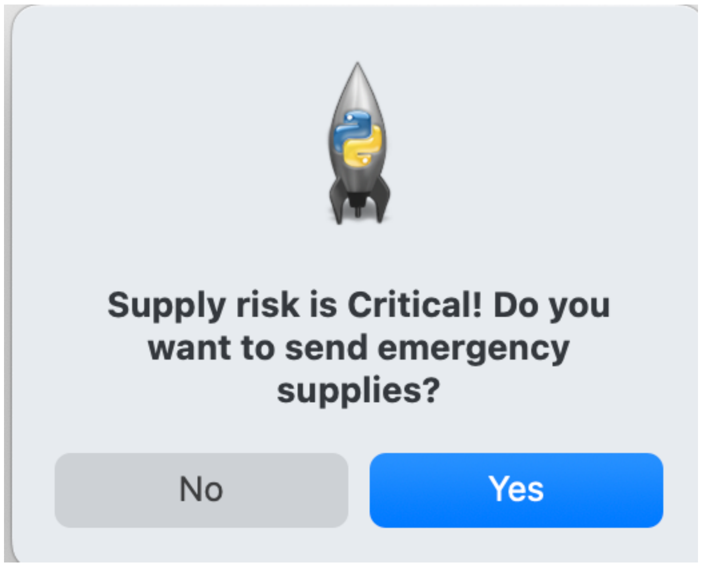
Enter troops for Southern Front (1-810):

23

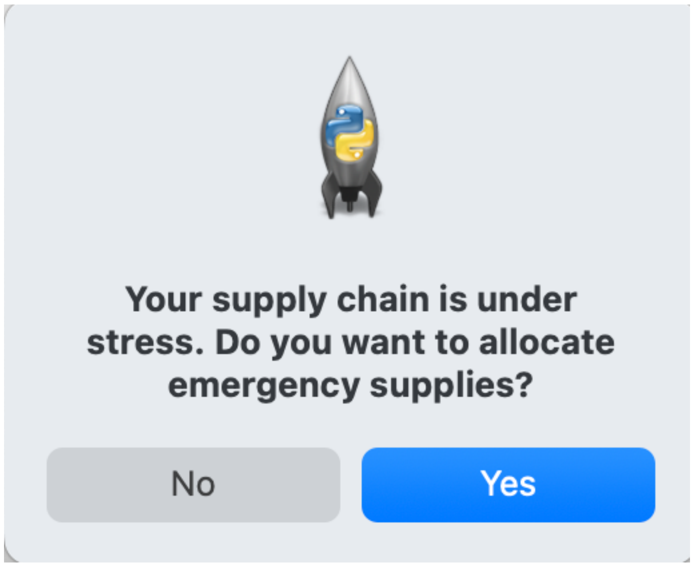
For testing purposes, I enter a random number in the 3 boxes, the python code stores the initial value of 1000 in the json file, which gets updated with usage, I reset it manually when testing. The point is, I input random numbers because in a world where mathematical models use AI tools, what I was doing here was that if I have a certain number of troops/ supplies, or whatever the factor may be, then the higher or lower number poses a greater or lower risk, and this would be done using an AI analysis of data, where if I had more time I could have implemented libraries which were built based on historical patterns. However, I still used my dataset and validated my data, with some margin of error of course, because after all, I only considered factors which can be analyzed statistically rather than the factors which may affect a battlefield, such as the allegiance of a country with others, etc. So the point of this part of the code is to have an idea, of a simulation of some numbers, backed up by historical trends, which affect the strategy of the battlefield, and furthermore, if in this initial step, I input numbers that my model thinks (with AI reasoning, which can be quite useful, but like with all models, only for guidance/ approximation), are too low, my code will display a warning that the risk is high, for example. So here is the next part:


You allocated troops across multiple battlefronts.

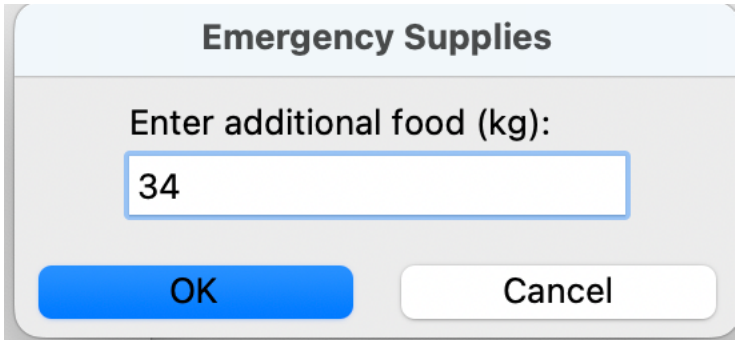
And then my model warns me that based on backend calculations, the risk is high:


Supply risk is Critical! Do you want to send emergency supplies?

And then the model will make some more prediction based on the sample numbers:


Your supply chain is under stress. Do you want to allocate emergency supplies?

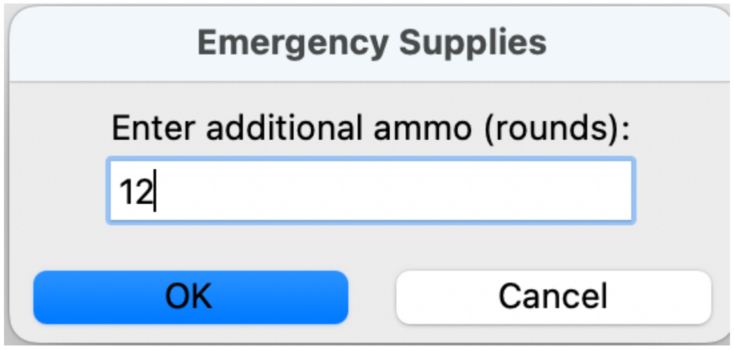
So it will advise me to add emergency supplies, when it thinks necessary:


Emergency Supplies

Enter additional food (kg):

34

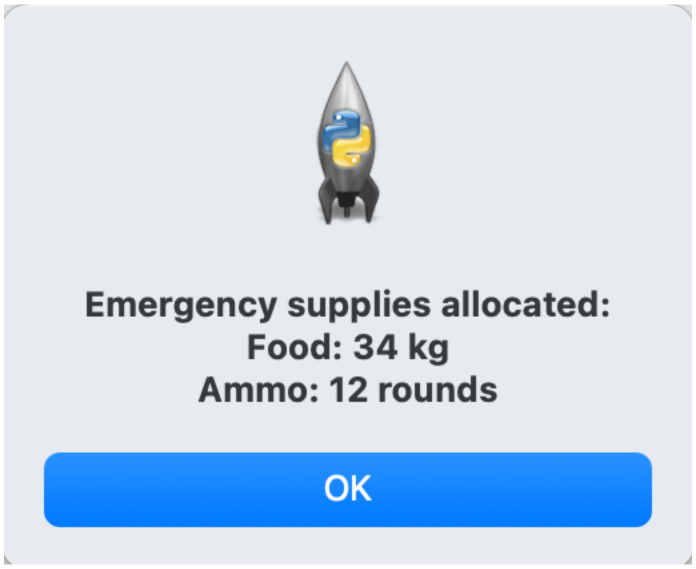
More supplies:


Emergency Supplies

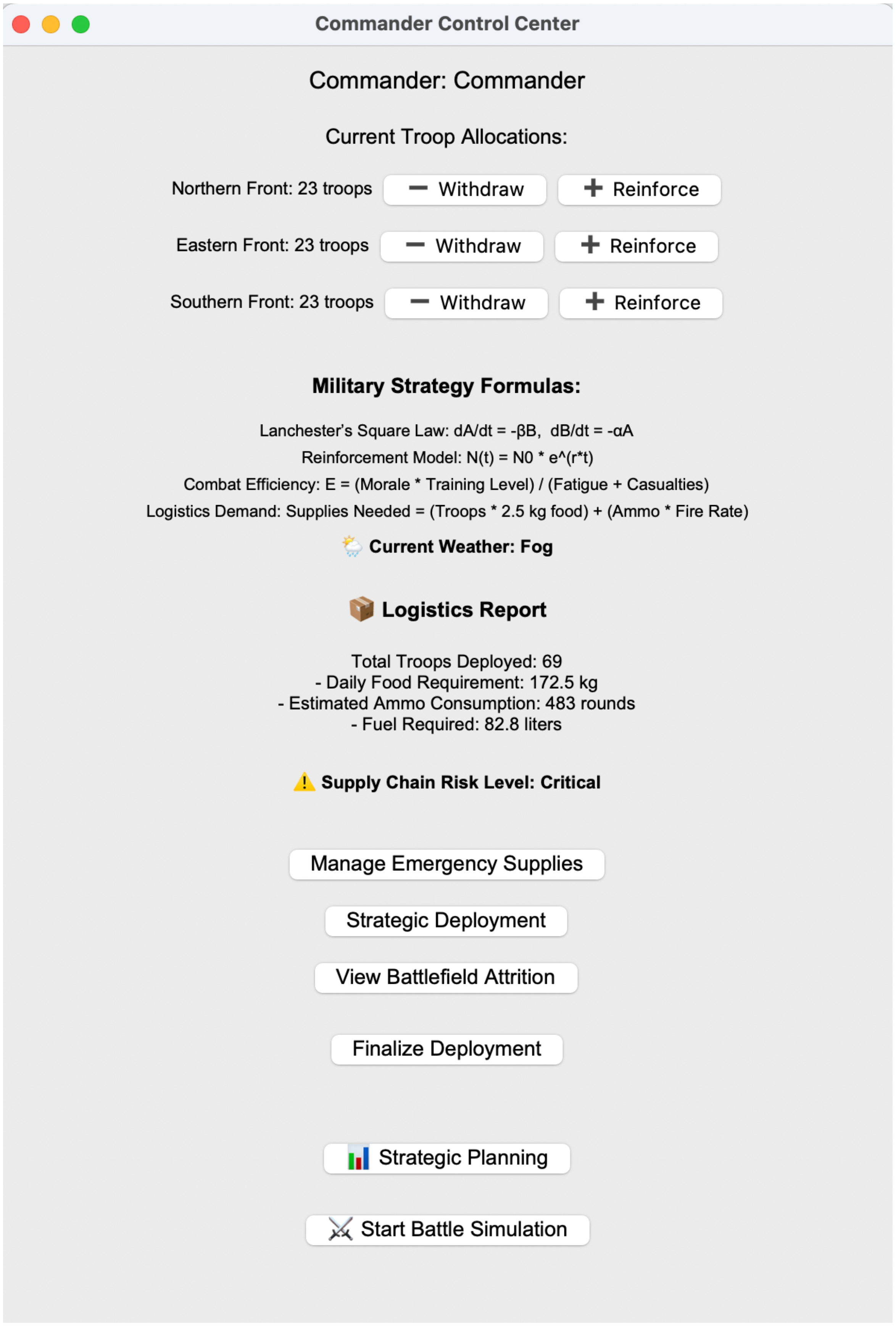
Enter additional ammo (rounds):

12

A summary:


Emergency supplies allocated:
Food: 34 kg
Ammo: 12 rounds

The main page:



Commander: Commander

Current Troop Allocations:

Northern Front: 23 troops

Eastern Front: 23 troops

Southern Front: 23 troops

Military Strategy Formulas:

Lanchester's Square Law: dA/dt = -βB, dB/dt = -αA

Reinforcement Model: N(t) = N0 * e^(r*t)

Combat Efficiency: E = (Morale * Training Level) / (Fatigue + Casualties)

Logistics Demand: Supplies Needed = (Troops * 2.5 kg food) + (Ammo * Fire Rate)

Current Weather: Fog

📦 Logistics Report

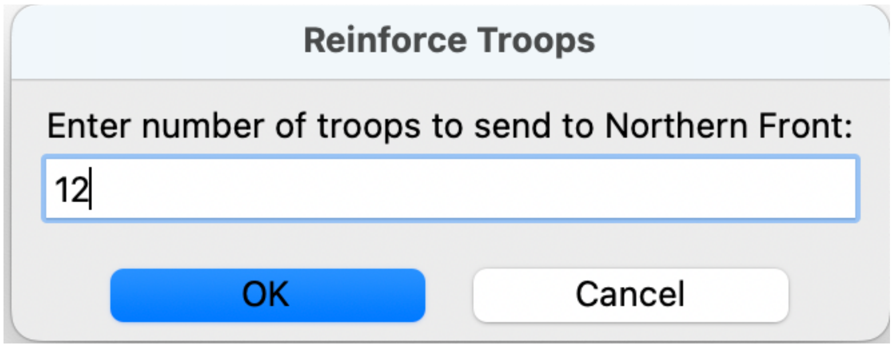
Total Troops Deployed: 69
- Daily Food Requirement: 172.5 kg
- Estimated Ammo Consumption: 483 rounds
- Fuel Required: 82.8 liters

⚠️ Supply Chain Risk Level: Critical

Manage Emergency Supplies
Strategic Deployment
View Battlefield Attrition
Finalize Deployment
📊 Strategic Planning
⚔️ Start Battle Simulation

The Commander Control Center is the main interactive window of the simulation, allowing users to manage troop allocations, evaluate battlefield conditions, and apply mathematical models in real-time. Troop deployments are divided across three fronts (Northern, Eastern, Southern), and each can be adjusted dynamically through Reinforce and Withdraw buttons. Below this, the interface displays a set of military strategy formulas that reflect the theoretical foundation of the simulation. These include Lanchester's Square Law, an exponential reinforcement model, a formula for combat efficiency, and a logistics demand function based on daily consumption rates. These formulas are not simply decorative, they are actively used in the logic that drives later simulation modules such as attrition and supply stress testing. A live weather condition (e.g., Fog) is shown to simulate environmental disruptions. The Logistics Report calculates food, ammunition, and fuel requirements based on current troop levels. A critical supply chain risk indicator is displayed when the system identifies excessive demands or vulnerable resupply routes.

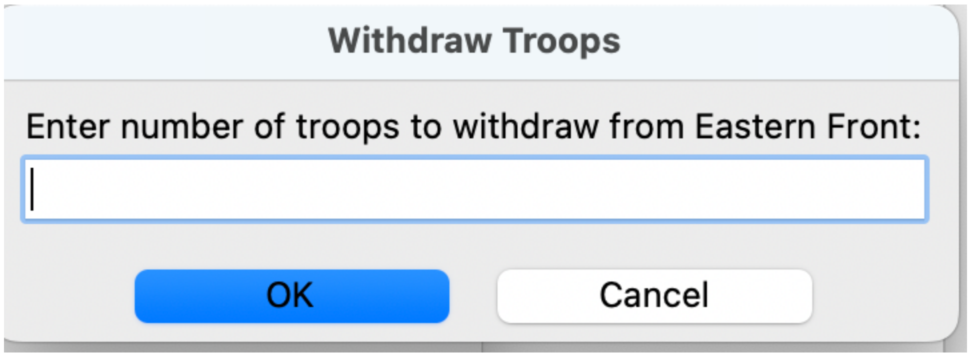
I start with the buttons at the top:


Reinforce Troops

Enter number of troops to send to Northern Front:

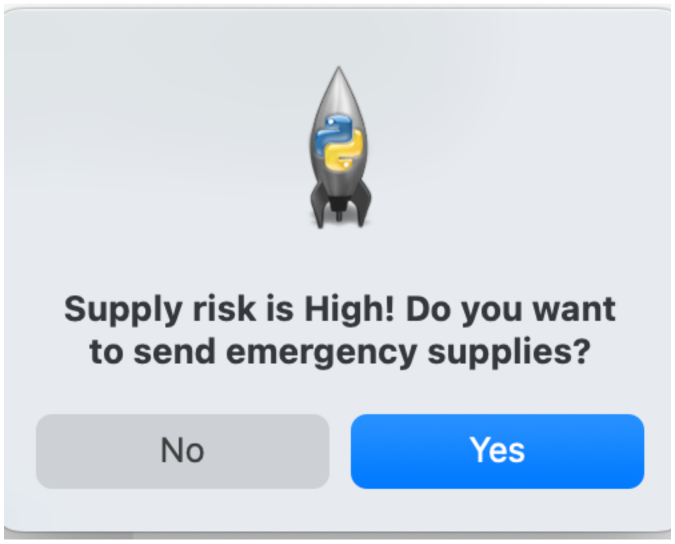
12

A sample number, but with more refinement, the model will be able to make predictions, advice based on the numbers:


Withdraw Troops

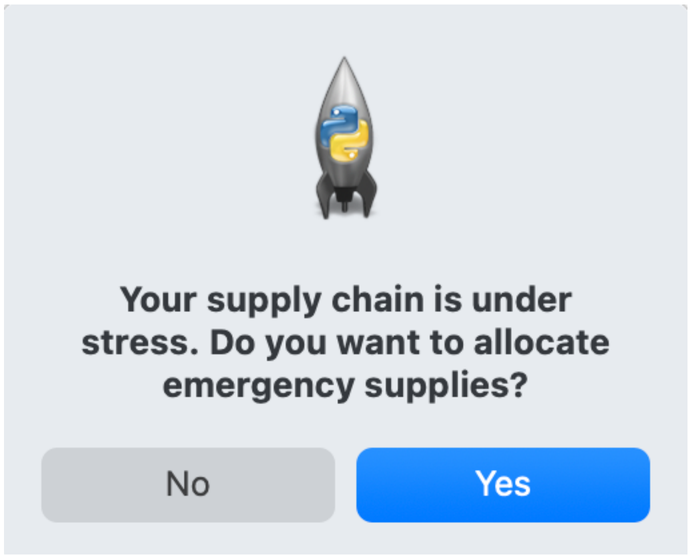
Enter number of troops to withdraw from Eastern Front:

Designed for a military stakeholder, but remembering a model by nature simplifies reality:



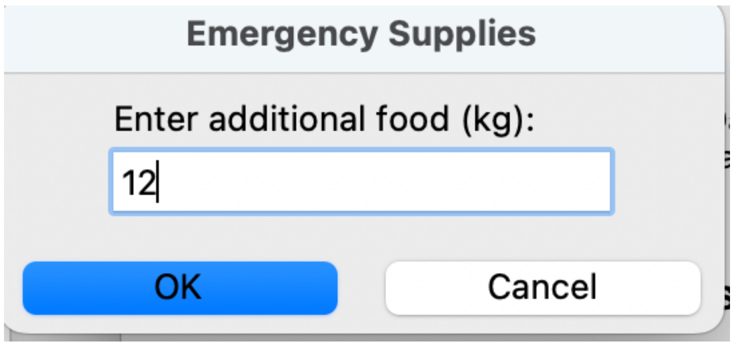
Supply risk is High! Do you want to send emergency supplies?

In fact, after withdrawing troops, my model thinks the risk is high, this is what I wanted to convey:



Your supply chain is under stress. Do you want to allocate emergency supplies?

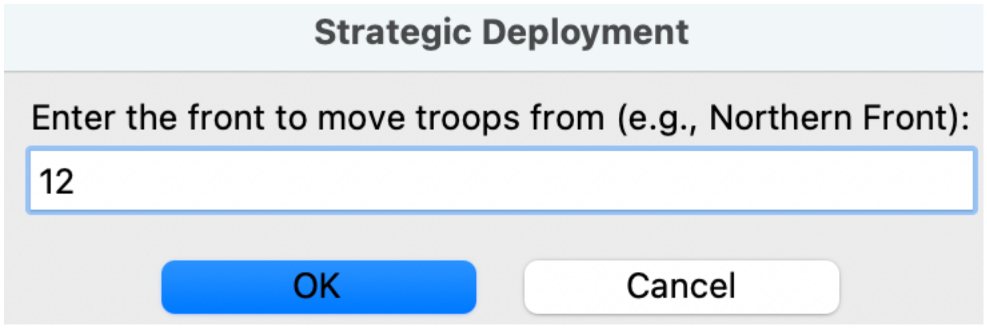
So I try to lower the risk, by adding supplies, which is one of the factors I looked into how the battle would be affected if there is a shortage (for example, if there is no food, the battle will be affected negatively because the soldiers cannot concentrate when they fight because they are starving):


Emergency Supplies

Enter additional food (kg):

12

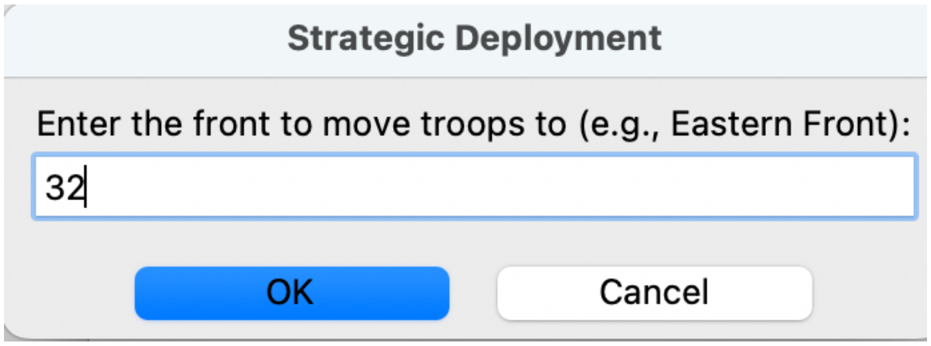
Next button:


Strategic Deployment

Enter the front to move troops from (e.g., Northern Front):

12

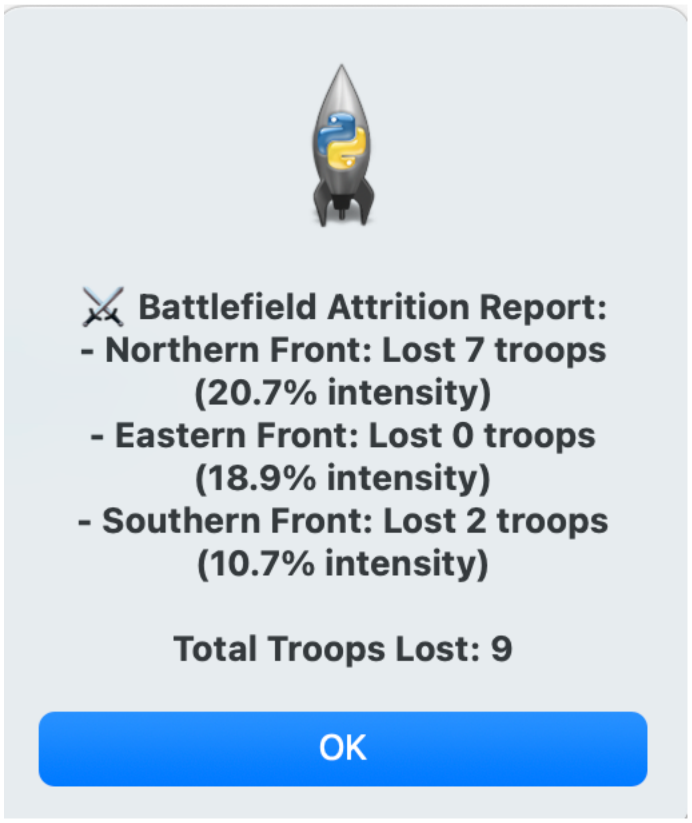
Then:


Strategic Deployment

Enter the front to move troops to (e.g., Eastern Front):

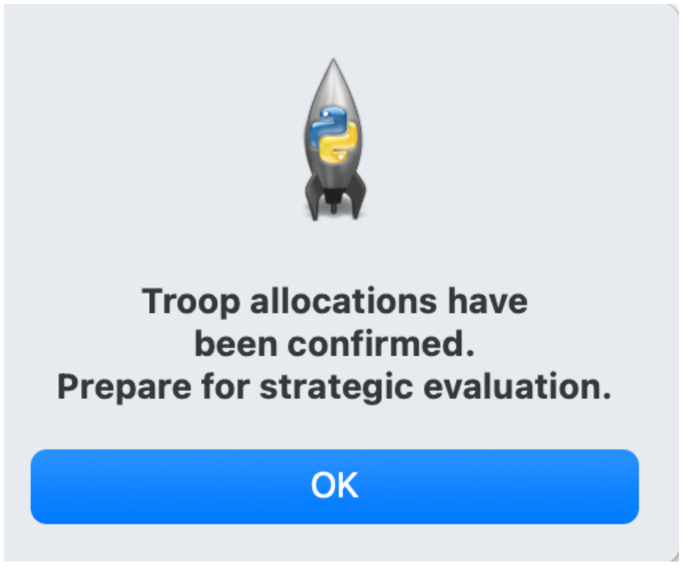
32

Battlefield attrition:



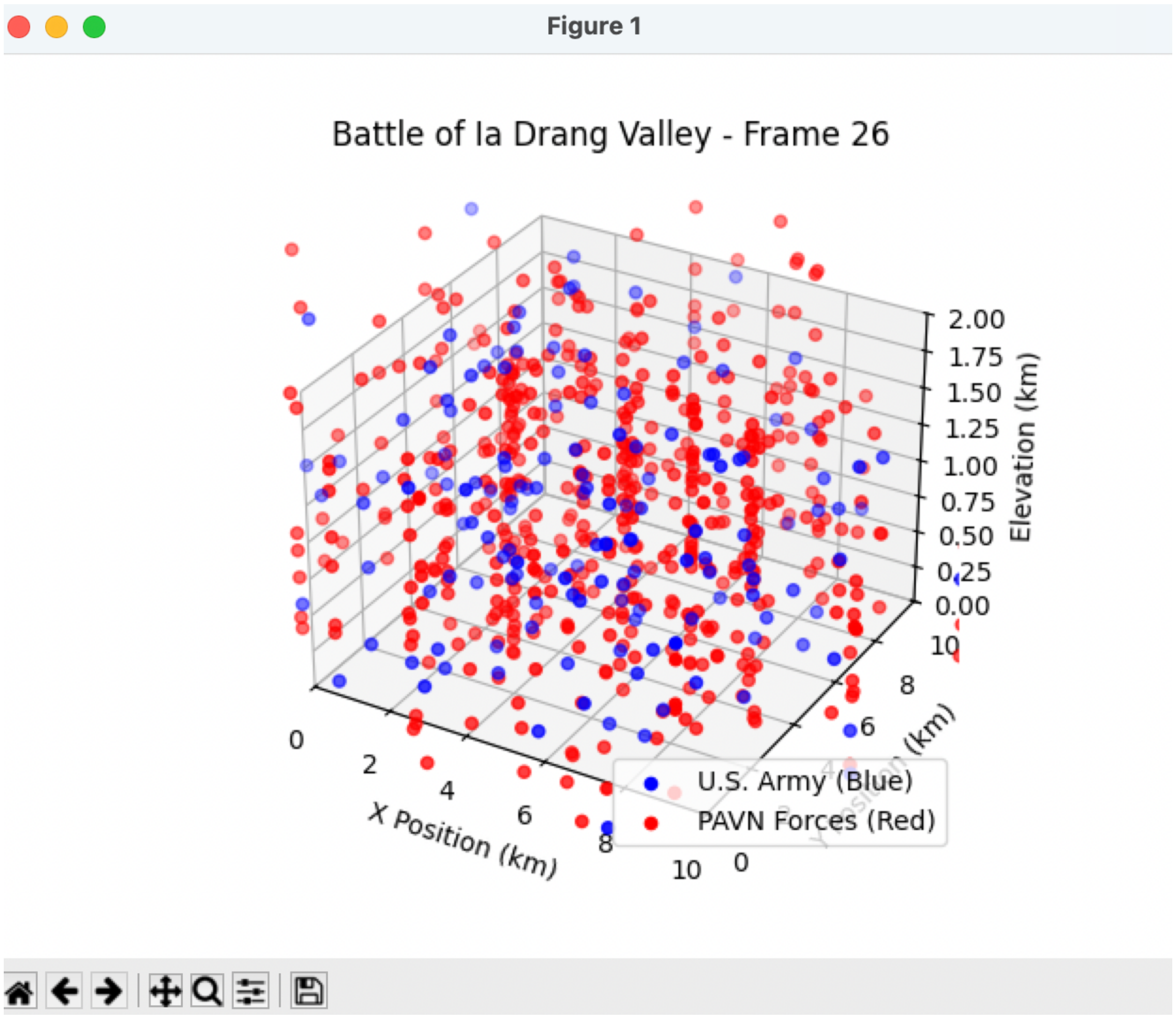
```
⚔️ Battlefield Attrition Report:
- Northern Front: Lost 7 troops
(20.7% intensity)
- Eastern Front: Lost 0 troops
(18.9% intensity)
- Southern Front: Lost 2 troops
(10.7% intensity)

Total Troops Lost: 9
```

A way to keep track of the battle, next thing:

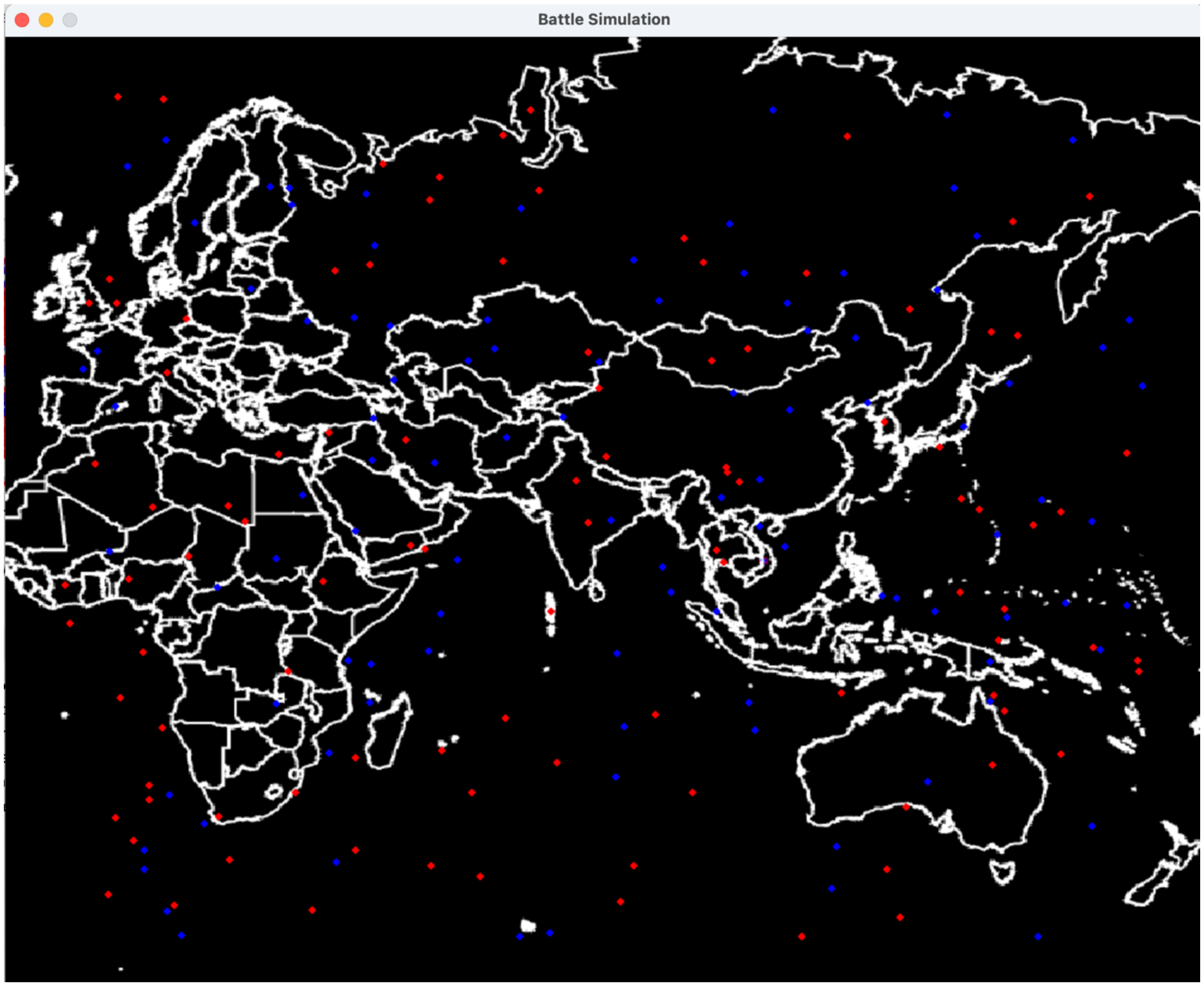


```
Troop allocations have
been confirmed.
Prepare for strategic evaluation.
```

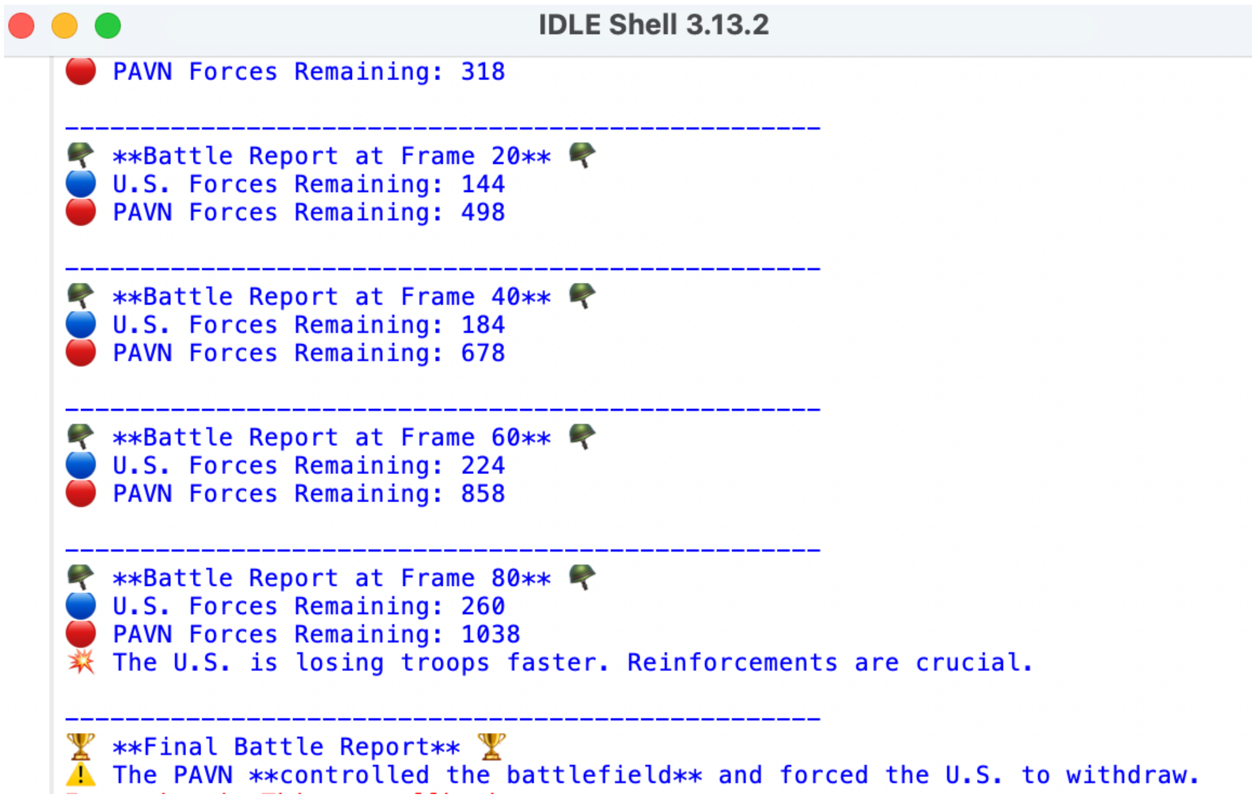
Due to a lack of time, some parts are incomplete, next:



A simulation in a 3d space predicting how the terrain, in this case elevation of the ground will affect the battle, using a simplified dots version to represent the soldiers. it is modelled as an xyz plane, where the blue represents u.s., if dots of the opposite colors collide, it means they are excluded from the frame (the frames are dynamic), the x axis corresponds to the blue army, the z axis the elevation (in this case, terrain), how the terrain (z axis in a simple R^3 coordinate plane, affects the chance of collision of the dots). Next thing:



This is certainly not finished; the result I wanted, but this shows the idea of the direction I was taking the project, and also inspired by the Mathematica notebooks from class, what I was trying to do here but didn't have time was to have the red and blue dots moving in the Vietnam part of the globe, in the areas where my dataset found there to be more casualties, which then I would cross reference with other datasets to tell me, in those regions, whether there was a higher or lower amount of supplies, and what the terrain was like.



```
🔴 PAVN Forces Remaining: 318

--------------------------------------------------
🪖 **Battle Report at Frame 20** 🪖
🔵 U.S. Forces Remaining: 144
🔴 PAVN Forces Remaining: 498

--------------------------------------------------
🪖 **Battle Report at Frame 40** 🪖
🔵 U.S. Forces Remaining: 184
🔴 PAVN Forces Remaining: 678

--------------------------------------------------
🪖 **Battle Report at Frame 60** 🪖
🔵 U.S. Forces Remaining: 224
🔴 PAVN Forces Remaining: 858

--------------------------------------------------
🪖 **Battle Report at Frame 80** 🪖
🔵 U.S. Forces Remaining: 260
🔴 PAVN Forces Remaining: 1038
💥 The U.S. is losing troops faster. Reinforcements are crucial.

--------------------------------------------------
🏆 **Final Battle Report** 🏆
⚠️ The PAVN **controlled the battlefield** and forced the U.S. to withdraw.
```

This screenshot illustrates the real-time output of the Python-based battle simulator developed as part of the project extension. The simulation reports dynamic troop counts for both U.S. and PAVN forces at successive time intervals (frames), reflecting the live attrition, reinforcement, and engagement logic defined in the backend. Each update provides a battlefield status report, quantifying remaining forces and flagging critical conditions, for instance, the U.S. suffering faster losses and requiring immediate reinforcements. The final report summarizes the outcome based on numerical superiority and sustained troop levels. While simplified, this mechanic showcases how mathematical modelling and AI based decision-making were integrated into a visual, interactive tool. It complements the Mathematica framework by transforming analytical patterns into live tactical simulations and user-facing feedback.