North Atlantic ’86

Summary

A fun game from the late 80’s that had a simple interface but complex game play. It captivated my attention 30 years ago. I spent countless nights playing and strategizing how to beat the game. I needed a reason to practice coding and re-creating this game was more fun than the various coding exercises. The requirements:

(1) Employ good coding practices
(2) The game should be playable
(3) capture in Github: https://github.com/sorens/north_atlantic_86

The game is broken down into four chief components:

(1) the data necessary to drive the game
(2) the game engine to prosecute the moves
(3) the user interface that allows the player to play the game
(4) the artificial intelligence engine to conduct the opponents moves

Each component will represent a separate phase of work. The first phase is to construct the classes needed to encapsulate the data of the game. The next task will be to construct the game engine, such that using automated mechanisms, games can be played even if they are just simple games (no UI, no AI). The third phase is to construct enough UI and UX to allow two players to play the game. The final phase will be to construct the AI engine so that a player could play a meaningful game against the computer.

Encapsulation and Data Abstraction

The encapsulation and data abstraction is an important foundation for the game. Creating the correct data abstraction will serve both the complexity of the subject matter while also satisfying the requirement to keep the interface simple. Each object (e.g. WeaponSystem, Unit, Fleet) will understand how produce the necessary data. The game object will encapsulate many of these various data abstractions together to allow the game to play. Each encapsulation will maintain strict boundaries and only work with the data it needs in order to do its job. I’ll use standard containers where appropriate to keep the implementation simple.

Game Engine

The game engine will leverage dependency injection and polymorphism so that the engine can be easily replaced or extended. A design goal for the game engine is to not allow any of the UI logic to leak into the design of the game engine. In other words, the game engine will receive data (the various moves by the players) and return data (the resolved combat and movement). The first version of the game engine will, as much as possible, reflect the original game play. This includes:

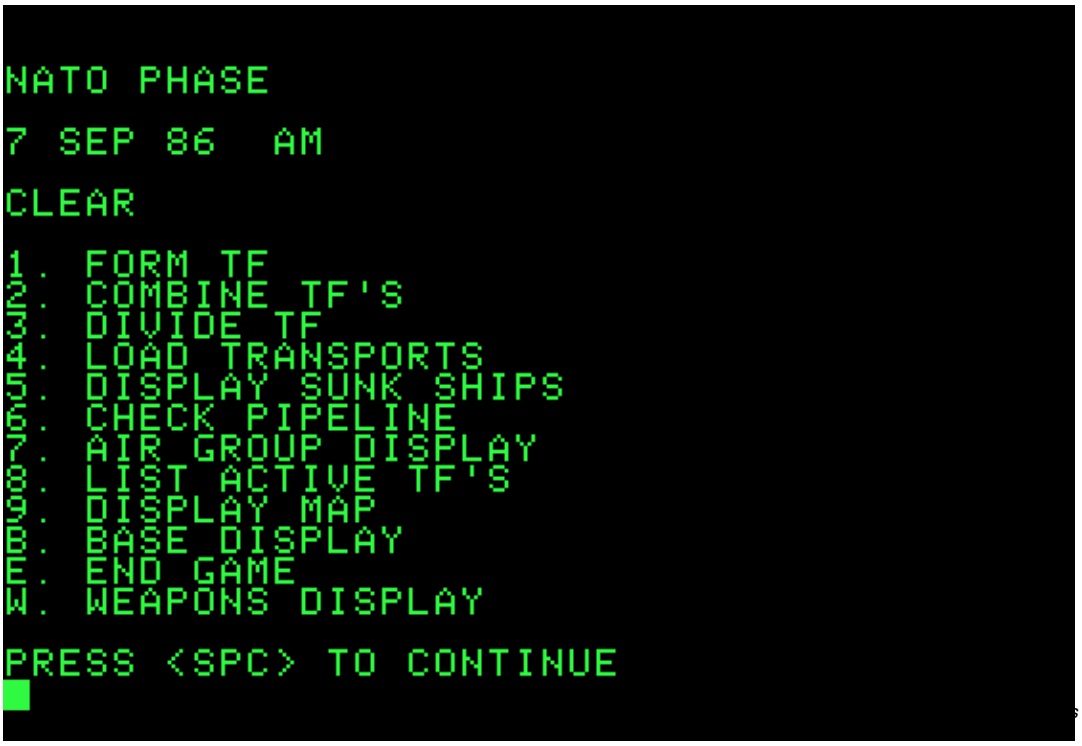
(1) A 12-hour turn for each player
(2) Each player takes one turn at a time
(3) Each turn consisted of:
    (1) Task Force (TF) Adjustment: Players could create, dissolve and edit TFs, load transports, display assets
    (2) Movement: TFs can be moved across the map during the Movement phase
    (3) Air Operations: Players could choose which assets were dedicated to combat air patrol (CAP) and which ones were dedicated to attack
    (4) Combat Resolution: During this phase, all combat would be resolved
(4) A stretch goal will be to serialize the data such that a game could be saved and restored

Later versions will increase the realism by adding or changing existing elements to augment the game. Areas of improvement could include:

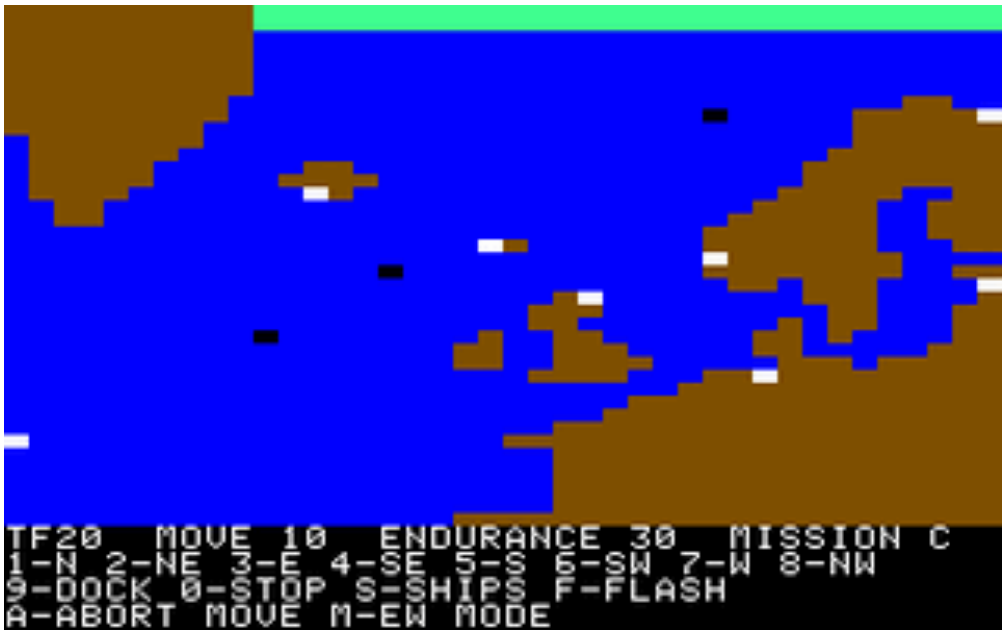
- Improved weather modeling (including improved detection degradation because of inclement weather)
- Detection and tracking via satellite
- Increase realism in detecting battlegroups in North Atlantic
- Search and cargo air assets subject to enemy CAP
- Improved control over Combat Air Patrol
- Formation editing and expected threat axis
- Modify the turn mechanism so that turns could "taken" simultaneously by each side before resolution.
- AI for the NATO player
- Map editor to allow other geographical areas
- New data using different ships and air assets

User Interface

The original game used a simple text menu interface to conduct the business of the game.



There was a graphical map that depicted where the units were located in the North Atlantic.



The first version of the game will pay homage to the initial game UI and UX. I will write a command-line engine to interpret commands and display the game as faithfully as possible to the original using the macOS console. In a later version, I will create a new interface for either a touch screen (iOS) or a mouse (macOS).

Artificial Intelligence Engine

The artificial engine has the most risk. While I understand the strategies and tactics that should be employed to win the game, writing that into maintainable and expandable code will require that I do more research on how to construct an AI Game engine. I have created games in the past, complete with computer players, but they were mostly brute force algorithms and decision trees. I am planning to do something similar for the first version, to encode a version of my strategies and tactics so I am playing against myself. However, I need to make sure that I do that in such a way that the engine can be swapped out for a better one. Keeping with the requirement that this game use good coding practices, the AI game engine will be dependency injected into the game object itself.

Automation and Testing

As each game component is constructed, a unit test mechanism will also be developed so that unit tests can be added and maintained through the process of creating this game. This will help prevent bugs which can occur once the code becomes complex enough that not all solutions can be calculated.