Increasing the hit to cost ratio of defensive kinetic warheads using a PPO agent in Unity

Ethan di Targiani
Mentored by Mr. Mikhail Wolff

Introduction

The gap in computational power between the human brain and computers has rapidly expanded. This gap has given artificial intelligence a chance to outperform the human brain in areas like defensive ballistic command. Machine learning (ML) is a type of artificial intelligence (AI) that imitates human learning using algorithms and models (Müller & Guido). This project used reinforcement learning (RL), a branch of ML, which is characterized by a reward system that guides the agent without requiring labeled inputs or outputs. The agent changes based on whether the reward value increased or decreased. An increase in reward signals the agent to adjust its policy and that this tree of decisions was overall beneficial.

The purpose of this project was to use reinforcement learning to create an agent in the game engine Unity3D to simulate the controlling of defensive kinetic warheads on maritime war vessels with the goal of decreasing the hit to cost ratio to 1:1.

Materials and Methods

To start the project, a baseline simulation environment was built in Unity with scripts created in C#. The ship, ocean, and missile models were created in Blender or imported from the Unity workshop. A buoyancy script was attached to the ship and the ship and missiles were placed into the ocean environment and given hitboxes. Missile flight patterns using artificial physics were created and interceptors were given a fixed speed. Flight patterns were modeled in 2D and later 3D space. An interceptor handler script was created to launch the six interceptors loaded onto the ship and a spawn zone was created in which the six threats randomly originated and were launched at the ship. To prevent the interceptors from being launched simultaneously, a launch period variable was used to set a time gap between interceptor launches. Finally, an agent handler script was constructed where a list of variables the agent could observe was compiled. The list of rewards was also stored in the agent handler.

The RL agent Proximal Policy Optimization (PPO) algorithm was used. The PPO algorithm is an RL neural network that updated the agent's policy. The learning rate (α) had a direct correlation with the amount of policy change, change to the algorithm. Epsilon (ε) dictated whether a policy change was too large, indicating the algorithm veered too far off course. Epsilon, learning rate, and values in the reward table were all adjusted throughout experimentation to optimize threat elimination (Table 1).

Parameters and Agents Table								
	A	B	C	D	E	F	G	H
α	0.0003	0.0003	0.0003	0.0003	0.05	0.05	0.05	0.0003
ε	0.2	0.2	0.2	0.2	0.2	0.3	0.2	0.2
r_1	−0.2	−0.2	−0.2	−0.2	−0.2	−0.5	−0.5	−0.2
r_2	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.2
r_3	−1.0	−1.0	−1.0	−1.0	−1.0	−1.2	−1.2	−1.0
r_4	0.6	0.6	0.6	0.6	0.6	1.2	1.2	0.6

Table 1: Table with each agent's parameters. Hyperparameters: learning rate ($0 < \alpha \leq 0.05$) and epsilon ($0.1 \leq \varepsilon \leq 0.3$). Reward table values: ship hit (r_1), threat neutralized (r_2), ship sunk (r_3), and all threats neutralized (r_4).

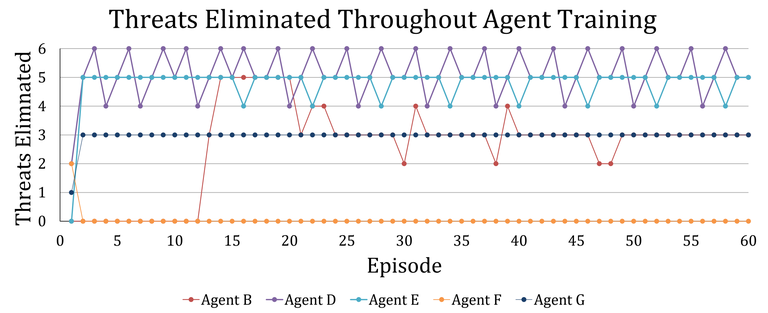
Eight iterations of the agent were trained and ordered from A to H in chronological order of creation. Each episode consisted of threats spawning randomly and the agent having to launch missiles in the correct order and time to protect the ship. During training, a Comma Separated Values creator script took the data and output the episode time, the number of threats eliminated, and the reward value for each episode into Excel for data analysis. Trends between threat elimination and parameters were found and recorded.

Results

The initial hypothesis was that an agent could be trained to decrease the hit to cost ratio to 1:1. Additional work is needed to improve consistency, but the PPO agent was able to eliminate more than three of the six spawned threats in three out of eight of the agents (Graph 1). Only Agent D was able to reach the hit to cost ratio of 1:1 highlighting the need for more testing and experimentation. Agents after Agent C had shorter launch periods between interceptors, which resulted in higher threat elimination. Changing the epsilon value from 0.2 to 0.3 had a negative effect on the agent's learning ability. A learning rate of 3×10^{-4} had a more positive impact on threat elimination than 5×10^{-2}.

Results (cont.)

Increasing the magnitude of the positive or negative reward values given to the agent had a negative impact on threat elimination. The ping-pong effect shown in some of the agents was due to the agent's inability to converge on a value during training, most likely caused by inefficient hyperparameters and simulation constraints such as the fixed speed of threats.



Graph 1: Plot of the number of threats eliminated each episode for each respective agent. Lines are connected to highlight trends in threat eliminations.

Conclusions

The purpose of the project was partially met. A PPO agent was created and trained in a Unity environment. Agent D was the most successful iteration and was able to destroy, on average, five of the six threats after multiple episodes of training. Further testing is required to consistently reach a hit to cost ratio of 1:1 and further exploration of the hyperparameters and reward table should be pursued. This study showed the applicability of reinforcement learning in defensive ballistics command and cemented AI's place in this field of study. The simulation environment could be used to train other types of RL agents, such as Q-learning, that were not explored during this study.

The study reinforced Unity, and other game engines, as a suitable workspace for simulating real-world scenarios. Using this study as a foundation the training ground could be expanded to other kinds of defensive applications.

References

Müller, A. C., & Guido, S. (2018). *Introduction to machine learning with python: A guide for data scientists*. O'Reilly Media, Inc.