Interceptor Missile Guidance through Deep Reinforcement Learning

Lance Fletcher , Roman Yoder

Abstract—The following project details the application of reinforcement learning to develop a policy for interceptor missile guidance. First the motivation and scope of the project are outlined. Then, the Markov Decision Process (MDP) is formulated and the related work is discussed. The problem is formalized as an agent missile tracking and intercepting a target missile before it has the chance to impact the ground. The implementation utilized an agent relative observation space and an advantage actor-critic algorithm. The reward function was uniquely developed and showed the convergence to desirable policies even in the presence of disturbances to the target missiles heading. Further, in some cases the agent could learn to account for states that resulted in a higher future probability of interception due to the approach characteristics.

- **GitHub:** https://github.com/LFletch1/Final-RL-Project
- **Youtube:** https://youtu.be/PxFW99PnIT0

Note: the contribution of each member is described in the appendix at the end.

Index Terms—Markov Decision Process, Reinforcement Learning, Missile Guidance

1 Introduction

Interceptor missiles are critical to military defense in the modern age of warfare by preventing incoming missiles from reaching their target. Interceptor missiles work by either directly colliding with a target missile or by detonating a small warhead to release shrapnel with the goal of detonating the target missile. Typically these are heat seeking missiles which are guided with a control system that utilizes infrared and radar sensors. Historically the guidance of a missile's trajectory was accomplished via traditional control theory; typically done through the use of a proportional navigation law. Traditional control methodologies have proved effective in the guidance of intercepting missiles but not without shortcomings [4]. These shortcomings result from the fact that implementation requires a dynamic model where even the most complicated of these models are reliant on simplifications and assumptions to predict the necessary control input. To combat this, the application of reinforcement learning (RL) algorithms to improve the overall performance has garnered recent attention. The motivation of this project is to propose, study, and implement the use of RL to guide interceptor missiles to target missiles.

2 Related Work

2.1 Reinforcement Learning in Missile Guidance

Sufficient guiding of a missile requires a model which encapsulates a continuous state space and action space. Shalumov [10] utilizes the policy gradient algorithm REINFORCE to develop a policy for a target-missile-defender engagement. In this situation a target missile defends itself from being intercepted by another missile. This is done by launching a defender missile which will intercept the other missile. The learned policy proved to successfully determine when the target missile should launch a defender missile to prevent interception. In *Anti-Interception Guidance for Hypersonic Glide Vehicle* [6], guidance of a glide vehicle is performed through developing a policy using a modified version of deep deterministic policy gradient (DDPG). Anti-interception guidance of a hypersonic glide vehicle has many real-time constraints similar to the guidance of a missile, thus policies created through RL are useful for many of the same reasons.

Some researchers have applied reinforcement learning to find the optimal gains of the missile guidance laws by utilizing traditional control theory to create a novel form of gain scheduling [5]. He *et al.* mentions that utilizing known information in dynamics and kinematics to structure a RL algorithm can improve efficiency based on prior knowledge. In fact, this approach of properly structuring the reward function and the Markov decision process (MDP) is not new and has been utilized by both He *et al.* and Du *et al.* Both take influence from traditional control error dynamics to structure their reward function and MDP. Rather than representing the state space of both the missile and target missile as Cartesian coordinates in the MDP state space, both authors take a more elegant approach, utilizing the relative distance and velocity from one missile to the other. Hu states that a naive and ineffective approach to the reward function is to simply base it on whether the missile successfully intercepts the target or not. Rather, a heuristic is developed for the reward function that is dependent on the zero effort miss (ZEM) and the constant reduction of the relative distance between the two missiles. ZEM can be defined as the geometric relationship to quantify the miss distance between the interceptor and target missile with the current velocity and a given interception time. [1], [4]. Both articles showed promising results, but an improved implementation by Hu *et al.* utilized the same MDP and

reward function but restricted the action output to that of the gain ratio in a traditional proportional navigation guidance control law with promising results.

2.2 Pursuit-Evasion Problems

A type of problem similar to the guidance of an interceptor missile is the pursuit-evasion problem. A pursuit-evasion problem involves an agent (pursuer) who chases another agent (evader) with the goal of catching it. It can easily be seen how the control of an interceptor missile has the same role as a pursuer in the pursuit-evasion problem. Thus, many techniques applied to pursuit-evasion problems are likely applicable to the specific missile guidance problem this paper focuses on. In [14], DDPG is utilized to train a pursuer in an environment where the pursuer and evader are modeled with the dynamics of cars. The paper also evaluates how well a pursuer and evader can learn simultaneously to learn behavior that helps their own opposing objectives. Similar to [3] and [5], the reward function is also based on the change in distance between the pursuer and evader. [13] improved on the multi-agent version of the pursuit-evasion problem by applying a multi-agent DDPG based on unmanned aerial vehicle dynamics. Along with the positions of each agent in the environment, individual agents also observe a set of sensors which allow them to detect objects and other agents in their vicinity. The reward function utilized is a combination of three reward functions based on the distance between a pursuer and evader, whether a pursuer has hit an obstacle, and whether the pursuer has successfully caught the evader.

The approach was influenced by many of the papers mentioned above, with many of our own modifications made to ultimately find a solution to the problem.

3 Approach

3.1 State Space

Throughout this paper the missile that will be controlled by a learned policy will be referred to as the agent missile and denoted as A. Also, the unit system used for distance and velocity is not specific to any specific measurement and are simply referred to as units. The missile which is to be intercepted will be referred to as the target missile and denoted as T. The overall implementation of the MDP and associated environment was more cumbersome than originally predicted. Through significant trial and error the MDP was modified while the reward function was fine tuned. Initially the observation given to the agent was simply the Cartesian coordinates and heading angle of both the agent missile and target missile, which proved to be a convoluted approach. A more effective approach that took influence from previous research [3], [5] was to define an agent centered state space. This reduced the overall complexity of the state-space and allowed the same relative configurations of the agent and target missiles to be the same regardless of global location. This was done by defining the state space as the distance, angle, and derivative of both quantities between the agent and target missiles. Additionally, the y-coordinate of the agent missile was included in the state space for the purpose of allowing the missile to know how close to the ground it is.

$$S = \{r, \lambda, \dot{r}, \dot{\lambda}, y_A\}$$

These state space variables were heavily influenced by [5] and can be seen in figure 1. Since this state space is relative to the agent missile, what would have been considered unique observations based on the global coordinates can be reduced, therefore different agent and target Cartesian coordinates combinations can be accounted as the same overall state by the agent. This simplified state space allows the agent to quickly learn an effective policy. A terminal state is when a collision between the agent and target missiles occurs. In this project, a collision is defined as when r is less than or equal to 20 units.

$$e = \{s : \forall S^+ \notin S\}$$

$$\text{s.t. } r \leq 20$$

The state is also considered terminal if either the target or agent missile impacts the ground.

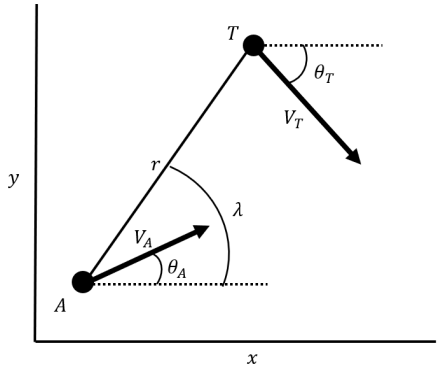

Fig. 1. Agent missile state space visualization. A very similar graphic appears in [5].

As shown in figure 1, r is defined as the relative distance between both missiles. While λ is the counter-clockwise angle of r with respect to the x-axis.

$$r = \sqrt{(T_x - A_x)^2 + (T_y - A_y)^2}$$

$$\lambda = \arctan\left(\frac{T_y - A_y}{T_x - A_x}\right)$$

Both $\dot{r}$ and $\dot{\lambda}$ can be defined as:

$$\dot{r} = V_T \cos(\theta_T - \lambda) - V_A \cos(\theta_A - \lambda)$$

$$\dot{\lambda} = \frac{1}{r}(V_T \sin(\theta_T - \lambda) - V_A \sin(\theta_A - \lambda))$$

Here V represents the velocity of the agent or target missile respectively and θ denotes the respective heading angle of the missiles with respect to the x-axis. Simply, θ represents the angle of velocity of a missile. Since these quantities can be defined analytically, there is no need to include past states in the observation space as done in many RL applications [8].

3.2 Action Space

Also necessary is the definition of the action space which represents the actions available for the agent to take. The variables and dimensions of the action space varied throughout the implementation of the project. Particularly it was debated whether the agent missile should be able to adjust its own velocity; however, for simplicity it was decided to only allow the agent missile to alter its heading angle. This action space can be defined below:

$$A(s) = \Delta\theta_A, \ \forall s \in S$$
$$\text{s.t. } -15^\circ \leq \Delta\theta_A \leq 15^\circ$$

3.3 Reward Function

Arguably the most critical aspect of the MDP to guarantee the convergence to a desirable policy is the design of the reward function. An initial naive approach was to reward the agent upon successful collision with the target missile and penalize it with a negative reward when the target missile impacted the ground. However, this does not result in a reward scheme that influences the agent missile to learn to steer towards the target. Another reward function we attempted was ZEM based and was very similar to reward function used in [5]. This reward scheme made use of the ZEM geometric constant as well as several other reward parameters, neglecting the use of an effort penalization.

$$R_t := R_z + R_v + R_r$$

The intention of this reward function was as follows: generate a negative reward any time the the relative missile distance grows with R_v, geometrically encourage the agent missile to steer such that the estimated future miss is decreased (which is scaled by its initial value) with R_z, and reward the agent when it is within a desirable radius of the target with R_r. For the environment used in this project, this reward function failed to converge, but some components of the function helped design a simpler and more robust reward function that resulted in the learning of an effective policy. The novel reward function utilized by our model is:

$$R_t = \frac{1}{r} - \dot{r} - 1$$

Where the inverse r and $\dot{r}$ portion of the function reward the agent for steering towards the target. The -1 in the function penalizes the agent missile every time-step; therefore influencing the agent missile to collide with the target missile quickly. Additionally, a large positive or negative reward was given to the agent when successful or unsuccessful termination states were reached respectively. The relative weight of inverse r and $\dot{r}$ components were tuned, as well as the rewards for the failure or success of the interception.

3.4 Reinforcment Learning Algorithm Utilized

Although the related literature suggested that the DDPG algorithm would provide desirable policies in this continuous action-state space environment it proved to be unsuccessful in this project's implementation. Initial project iterations found success with a discrete action space and the use of a Deep Q-Network (DQN). However, to handle the continuous action space, advantage actor critic (A2C) proved to be a viable solution.

A2C relies on both an actor and a critic network, and can be thought of as a policy gradient method. Policy gradient methods can be easily amended to continuous action spaces as the DNN utilized to output the given action associated with a provided state can output a probability distribution. A2C makes use of ideas from REINFORCE in that it updates the given policy proportional to an advantage, or return in the case of REINFORCE, in the direction of this return but inversely proportional to the probability of this return. This is advantageous as the policy is not superfluously shifted towards commonly occurring actions that have been biased with a higher probability. However, unlike REINFORCE, A2C can perform online and incremental learning as the RL algorithm can make use of n-step temporal difference learning through the use of a critic- rather than REINFORCES Monte Carlo approach. Overall actor-critic approaches can be best thought of with the one-step advantage actor critic update [7], [12]:

$$\theta_{t+1} = \theta + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\frac{\nabla\pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)}$$

Here the critic can be thought of as a DNN to approximate the state-value function $\hat{v}(S, w)$ and is parameterized by the vector w. The advantage can be thought of as the net gain or loss from taking an action at a state, as such the parameters, θ, of the policy function $\pi(A|S, \theta)$ are updated in the direction of increasing this actions probability scaled by the advantage. The critic is also updated utilizing the advantage function which adjusts the parameters according to the received reward at time step t, the estimation of the next states state-value and the estimation of the current states state-value:

$$w \leftarrow w + \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\nabla\hat{v}(S_t, w)$$

Clearly if a higher (or lower) than expected reward is received the critic will adjust the parameters w to compensate for the discrepancy. This one step A2C can easily be modified to utilize continual updates via eligibility traces. Although the use of the critic introduce some bias, more prevalent in one-step, by relying on the localized estimation of the critic; this is easily made up for with the reduction of variance. This variance no longer results due to the bootstrapping of the critic, computing each advantage according to the perceived reward and the localized state-value estimates [12].

4 Implementation

4.1 Frameworks

The environment was built using the OpenAI Gym framework [2]. A graphical visualizations of the missile trajectories was built by using Pygame [11] (See appendix). The A2C algorithm implemented by Stable-Baselines3 [9] was used to quickly train a variety of RL models.

4.2 Varying Target Missile Behavior

To thoroughly test the proposed method and explore its capabilities target missiles were tested of varying types as described in the table below. A policy for each missile type was learned independently by using episodes specific to the target behavior. Each target missile type was trained for one-million time steps or roughly 3500 episodes. Each episode the agent missile started in the same position (lower left portion of the environment) and the target missile was placed in a random position at the top of the environment. The target missile's trajectory was guaranteed to hit within a certain range of value along the ground. This was done to prevent the target missile from flying off the screen of the visual interface.

Target Type	Description
static	A stationary target missile. Not realistic, but was primarily used for prototyping of the reward function.
dynamic	A moving target missile with the same velocity every episode.
v_dynamic	A moving target missile with a potentially different velocity every episode.
noisy	A moving target with random alterations to its velocity angle occurring throughout the episode. Same velocity every episode.
all	A moving target with random alterations to its velocity angle and potentially different velocities every episode.

4.3 Hyperparameters

There were various hyperparameters specific to A2C that were tuned throughout the project implementation. The values in the table below proved to be successful, but further hyperparameter tuning could generate better results. Also, any hyperparameters not specified were the default values provided by the Stable-Baselines3 framework [9].

Hyperparameter	Value
Learning Rate	7×10^{-4}
Entropy Coefficient	10^{-2}
N steps per update	10^{3}
Discount	0.99

5 Results

Once each policy had been trained for each type of target missile, the performance of each policy with its respective missile type was tested. As seen in figure 2, the model was able to learn a policy which could consistently intercept target missiles of the types static, dynamic, and noisy. The model failed to learn desirable policies for the v_dynamic and all target missile types. Since the velocities potentially change between episodes, the agent missile fails to learn a policy which can account for different target missile velocities. Additionally, the agent missile likely struggled to collide with the target missile because of the target missile's velocity being too large. If the maximum velocity of the target missile was decreased, the agent missile might have shown better performance.

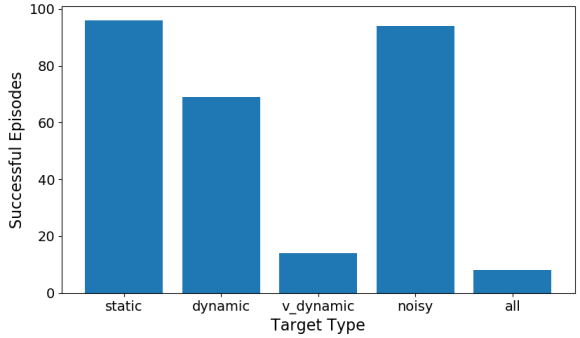

Fig. 2. Number of successful episodes out of 100 for various missile types.

For the noisy target missile, the agent converged on a policy which would attempt to circle around the target missile, and then hit it from the rear as seen in figure 3. The sampled trajectories from the trained policy in this noisy environment demonstrated the policie's robust ability to handle disturbances in the heading of the target missile. Many real world missile interceptors take this approach; however, it is unclear if this policy was converged upon because it optimizes the chance of interception, or if the environment set up is biased towards this type of trajectory.

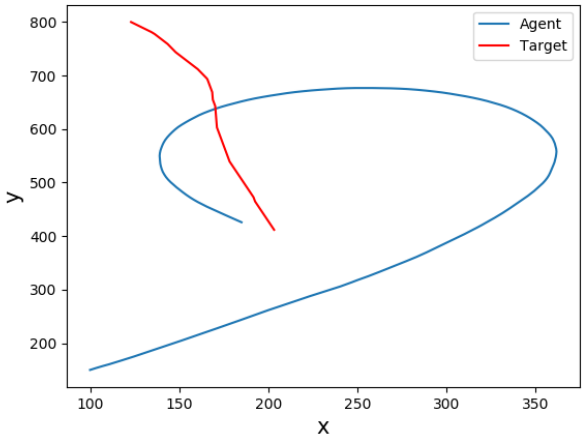

Fig. 3. Trajectory taken by agent missile with a noisy target missile. See appendix for additional trajectories.

In figure 5, it can be seen that for the noisy target missile the episode rewards increased as the training progressed. The policy converged relatively quickly at around episode 1000. For the episode length, a less clear trend is plotted in figure 4. The episode length on average did decrease, which shows the agent missile learning to hit the target missile sooner to maximize rewards. Throughout the plot there are a few episodes which reached the episode length limit of 1000. This is likely due to the noisy missile's trajectory being altered enough that an uncommon observation is provided

to the agent missile and thus the policy does not perform as well.

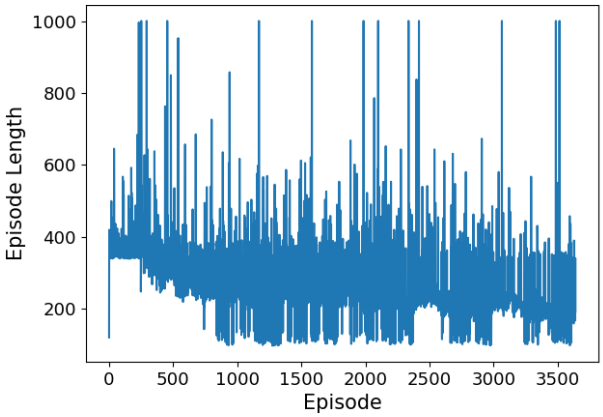

Fig. 4. Episode lengths as model trained with noisy target missile.

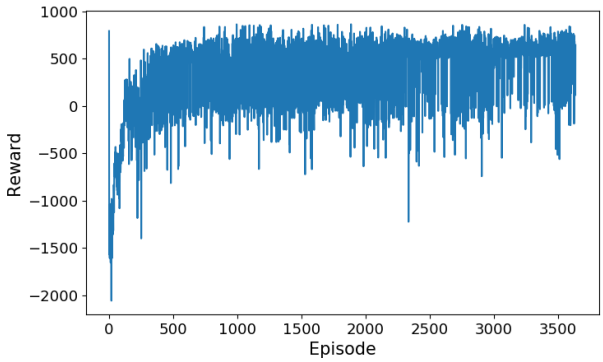

Fig. 5. Episode rewards as model trained with noisy target missile.

6 Discussion

When comparing the different target missile types it is clear that the agent missile struggles to intercept the missiles which vary in speed each episode. A potential solution to this issue would be to allow the agent missile to also control its velocity. The results of implementing this seemed to show the model needed significantly more training. For the sake of time, this was not included this in the final implementation.

Frequently upon convergence, the policy is found to drive the agent missile toward the target missile in an opposing direction only to then circle around the target missile and intercept it from behind. This could be a coincidence due to the lower probability of successful head-on interception but could also be a learned behavior. When trained with a constant velocity target missile the agent may have learned to associate which values of $\dot{r}$ are associated with head-on approach and learned to avoid these opting for a lower $\dot{r}$ associated reward but a higher probability of a positive terminal reward. This also can likely be attributed to the poor performance of the v_Dynamic environment as randomizing the target missiles velocity impeded the agents ability to from associations between the values of $\dot{r}$ and approach condition.

7 Conclusion

The method used proved to be successful with a variety of target missile types. The novel reward function allowed the model to learn a policy capable of consistently intercepting the target missile. Of course the methods used could be improved. As discussed earlier, adding control of the agent's velocity to its action space could yield better results for certain target types. The lack of physics utilized in the environment creates a significant reality gap; however, the environment still showed the agent's ability to make advantageous trajectory decisions. Further, other RL algorithms such as DDPG and A3C could be experimented with in the future. Work should be done on applying similar techniques in 3-Dimensional space while preserving the developed nuances in the reward function; although, this would likely require far more computational resources and training iterations. Even with RL, real world missile guidance likely needs some amount of traditional guidance control. The coupling of both methodologies could lead to promising results. Even with some minor shortcomings this project proved to be an excellent exploration into RL algorithms being applied to a real world problem.

References

[1] Ben Dickinson. Zero Effort Miss - Section 4 Module 3 - Missile Guidance Fundamentals, February 2021.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. _eprint: arXiv:1606.01540.

[3] Mingjian Du, Chi Peng, and Jianjun Ma. Deep reinforcement learning based missile guidance law design for maneuvering target interception. In *2021 40th Chinese Control Conference (CCC)*, pages 3733–3738, July 2021. ISSN: 1934-1768.

[4] Shaoming He and Chang-Hun Lee. Optimality of Error Dynamics in Missile Guidance Problems. *Journal of Guidance, Control, and Dynamics*, 41(7):1624–1633, July 2018. Publisher: American Institute of Aeronautics and Astronautics.

[5] Shaoming He, Hyo-Sang Shin, and Antonios Tsourdos. Computational Missile Guidance: A Deep Reinforcement Learning Approach. *Journal of Aerospace Information Systems*, 18(8):571–582, August 2021.

[6] Liang Jiang, Ying Nan, Yu Zhang, and Zhihan Li. Anti-Interception Guidance for Hypersonic Glide Vehicle: A Deep Reinforcement Learning Approach. *Aerospace*, 9(8), 2022.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. arXiv:1602.01783 [cs].

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. arXiv:1312.5602 [cs].

[9] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[10] Vitaly Shalumov. Cooperative online Guide-Launch-Guide policy in a target-missile-defender engagement using deep reinforcement learning. *Aerospace Science and Technology*, 104:105996, September 2020.

[11] Pete Shinners. PyGame, 2011.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[13] Kaifang Wan, Dingwei Wu, Yiwei Zhai, Bo Li, Xiaoguang Gao, and Zijian Hu. An Improved Approach towards Multi-Agent Pursuit-Evasion Game Decision-Making Using Deep Reinforcement Learning. *Entropy (Basel, Switzerland)*, 23(11):1433, October 2021.

[14] Maolin Wang, Lixin Wang, and Ting Yue. An Application of Continuous Deep Reinforcement Learning Approach to Pursuit-Evasion Differential Game. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1150–1156, March 2019.

APPENDIX

Figures

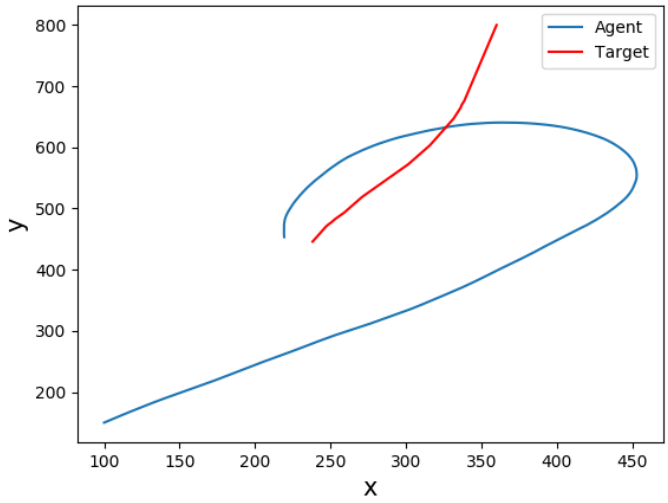

Fig. 6. Trajectory

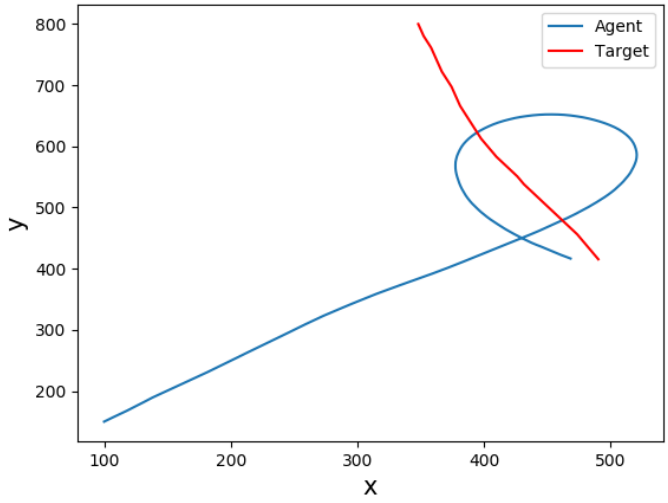

Fig. 7. Trajectory

Fig. 8. Pygame Environment

Contributions

Both team members equally contributed to the project in their own respect. Lance can be contributed with developing the python environment and successfully integrated stable-baselines. Roman aided heavily in the conceptual development of the MDP and reward function- as well as debugging and confirming that the observation space was appropriately calculated. The majority of the project was done together in person and was a collaborative effort with both members benefiting off each other's experience and opinions.