Pursuit-evasion game with SARSA learned pursuer

Author:
Thomas Mutsaers
5867711
t.mutsaers@student.uu.nl

Bachelor Thesis Artificial Intelligence
Utrecht University
7.5 ect

Supervisor:
Natasha Alechina

Second reader:
Tejaswini Deoskar

13 November 2020

Abstract

Pursuit-evasion algorithms have become more important to domains in robotics, like surveillance with robots. Most of these domains lack a complete model or enough data to work with, reinforcement learning could be a good solution to this. This thesis will show that "a pursuer with a Sarsa algorithm is able to catch an evader in a discrete grid environment." First, pursuit-evasion games and Sarsa algorithms will be explained. The algorithm used in this thesis implements a simple two-dimensional environment with obstacles in which a pursuer agent tries to catch an evader agent. In the experiments pursuer uses a Sarsa algorithm with different parameter settings against a evader that used two different behaviours. This algorithm will be described further in the method section. The pursuer manages to learn to catch the evader in each tested scenario. There is not a very noticeable difference between the different parameter values used for Sarsa, and there is a difference between the two behaviours of the evader. Finally, there will be discussed what would be interesting for future research. The code used for this thesis can be accessed here: https://github.com/tts118/sarsa-pursuit-evasion .

Contents

1. Introduction 3

2. Theoretical Background 3

2.1 Pursuit-evasion games 3
2.2 Reinforcement learning 3
2.3 SARSA 4
2.4 SARSA compared to Q-learning 5

3. Method 5

4. Results 6

5. Conclusion 7

6. References 8

7. Appendix 9

1. Introduction

Pursuit-evasion algorithms have become more important to many domains in robotics. Examples of applications where these algorithm can be relevant are controlling unmanned drones, following targets, surveillance with robots and emergency response when looking for a lost person. [2] Since its applicable in so many different areas which do not all have a complete model or enough data to work with, reinforcement learning could be a good solution to this. [1] uses reinforcement learning for a drone to detect and follow another drone. However few researches have used Sarsa to solve a pursuit-evasion game. This thesis will show that “a pursuer learning with a Sarsa algorithm is able to catch an evader in a discrete grid environment”.
The next section will explain what pursuit-evasion games are and how reinforcement learning and Sarsa are relevant to this and how they work. Section 3 will explain then environment and how Sarsa is implemented in it. After that section 4 will give an overview of the results. In section 5 a conclusion will be made, the result will be summarised and recommendations for future research will be given.

2. Theoretical Background

2.1 Pursuit-evasion games

Pursuit-evasion games have one or more pursuers trying to catch one or more evaders. [2, 3] These games are used to study path planning, where the pursuer(s) try to catch, find or follow the evader(s). Examples of such scenarios are playing hide-and-seek, finding a lost person or catching a robber. Because of the large number of different situations pursuit-evasion games are applicable, there are many variations on pursuer-evasion games. Environments can vary from continuous planes to graphs or grids and many more. The amount of information available to each agent at any time, like the location of the other agent, might be limited or completely open. The manoeuvrability of each agent can be different in both speed and available direction and the definition of capture can vary.

The goal with pursuit-evasion games is often to find the optimal path for the pursuer(s) in order to catch the evader(s) or to determine the minimum required amount of pursuers needed in a certain scenario in order to guarantee the capture of the evader(s). This can be solved in different ways, in this thesis reinforcement learning will be used. A reason to use reinforcement learning is that it has no need for a complete model nor does it have a need for training data.

2.2 Reinforcement learning

Reinforcement learning is a way to understand and automate goal-directed learning and decision-making. It differs from other machine learning methods by having an agent learn from direct interactions with its environment, without needing any supervision or complete models of its environment. [4] This makes reinforcement learning useful for settings where it is difficult to make a complete model of the environment and when there

is not enough training data. Most pursuit-evasion games that use reinforcement learning use a variation of Q-learning [3] combination with deep reinforcement learning [1], but there are very few using Sarsa.

2.3 SARSA

SARSA is an on-policy, model-free reinforcement learning algorithm. The explanation of the algorithm is based on [4]. A model is a representation of the behaviour of the environment. Making an accurate model often requires a considerable amount of knowledge and understanding of the environment's behaviour. The advantage of a model-free reinforcement learning algorithm is that there is no need for such a model. In a model-free reinforcement learning algorithm the agent relies on trial-and-error to update its knowledge. A policy, in the case of sarsa, is a mapping from all states-action pairs in the environment to an action to be taken when in those states. Here a state is the current situation and can vary highly depending on the goal of the reinforcement learning algorithm. This can be the location of the agent self and the last know location of the evader for this experiment or the position of the legs of a robot that is learning to walk. The action is what the agent can do in a state. The goal of reinforcement learning is to find the optimal policy, which is the policy with the optimal action for each state. Sarsa is an on-policy because the agent updates its policy based on the current action obtained from the current policy, as opposed to off-policy where the agent updates its policy based on the optimal action based on a different policy.
Sarsa uses $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$ as its update function for the policy. This function will be used after every action the agent takes. Here $Q(s_t, a_t)$ is the policy value for the state s_t when doing action a_t. $r_{t+1}$is the reward that the agent earns for the taking the action a_t in state s_t . s_{t+1} is the new state that the agent will be in and a_{t+1} is the action the agent is taking in that state according to the current policy. α is the step-size parameter of the agent, a number between 0 and 1, and influences the learning rate. If α is closer to 1 the new information, $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$, will contribute a lot to the updated value of $Q(s_t, a_t)$. When α is closer to 0 the new information will barely contribute to the updating of the value of $Q(s_t, a_t)$, and when α is 0 the policy will not be updated and thus not learn. γ is the discount rate is a number between 0 and 1 which determines the present value of future rewards, $Q(s_{t+1}, a_{t+1})$ in the function. When this value is closer to 1 it will value future rewards highly and at 1 just it will value obtaining rewards in the future as highly as obtaining them at the moment. A discount rate closer to 0 does not value future rewards much, when it is 0 it will only value immediate rewards and no future rewards.
In order to incentivise the agent to explore it will be given an exploration rate ε between 0, only exploration, and 1, no exploration. When the agent chooses an action it will take the best action given the policy or a random action, depending on epsilon. *Figure 1* shows the pseudocode of sarsa.

```
Initialize Q(s,a), for all s ∈ S, a ∈ A(s), arbitrarily, and Q(terminal-state, ·) = 0
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        Q(S,A) ← Q(S,A) + α[R + γQ(S',A') − Q(S,A)]
        S ← S'; A ← A';
    until S is terminal
```

Figure 1: Sarsa pseudocode [4]

2.4 SARSA compared to Q-learning

Q-learning is an off-policy, model-free reinforcement learning algorithm. The explanation of this algorithm is based on [4]. As an off-policy reinforcement learning algorithm, q-learning differs from sarsa in that it updates its policy based on the optimal action chosen from a policy independent from the policy that is followed. The reason for picking sarsa over q-learning is that sarsa is more conservative than q-learning. This is in most simulations not very relevant, but when training with, for example, robots that are costly, it is better for the robot to learn safely instead of endangering itself. So by experimenting if sarsa can solve the pursuit-evasion problem insight can be gained on whether or not it's an option for training robots.

3. Method

The environment for this experiment is a 10x10 grid. On this grid there are four predetermined walls. *(figure 2)* On this grid two agents will be randomly placed on empty tiles. One agent is the pursuer, which will be trained using the SARSA algorithm, and the second agent is the target, which will use a predetermined strategy. The pursuer's goal is to learn to catch the target as quick as possible. The target will try to move as far away as possible from the pursuer while it has line of sight, when the target does not have line of sight to the pursuer it has a chance to act randomly or still move away from the pursuers last known location. The target it considered caught when the pursuer and the target occupy the same tile. Each agent has four different actions it can execute, it can either move north, west, south or east. If the chosen action would results in moving of the map or into a wall, the agent will not move. Both agents will take their actions simultaneously. The agents will both know the starting location of the other agent. During each step it is checked if the pursuer and the target have line of sight of each other, if this is the case the last known location of the other agent will be updated, otherwise they will only know the last known location of the other agent.

The state space used for SARSA's q-table is $[x_p, y_p, x_t, y_t, l, a]$ where (x_p, y_p) is the location of the pursuer, (x_t, y_t) is the location of the target. $l = 0$ means that the pursuer and target have no line of sight when $l = 1$ they do, and a is the action the pursuer is planning to take with $0, 1, 2, 3$ being north, east, south and west respectively. All states are initialised to a

random value between 0 and -1, apart from the end states, which are initialised to 0. The reward for each step the pursuer takes is -1, unless the step results in reaching an end state, than the reward is 10. Each episode is maximum 200 steps long, since a lower number of maximum step did not allow the agent to learn properly (figure 11) and a higher maximum does not seem to improve the learning.

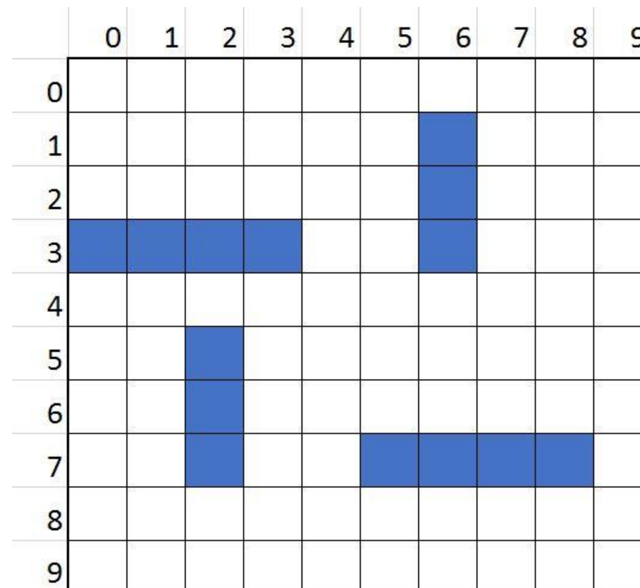

Figure 2: Environment with walls and coordinate numbers

4. Results

Figure 3 shows the total reward the pursuer obtains per episode with $\alpha = 0.95$, $\gamma = 0.95$, and $\varepsilon = 0.85$. A reward of -200 means that the pursuer did not manage to catch the target, for all other values the pursuer did manage to catch the target. A higher value means a faster catch by the pursuer. It is clear that the pursuer learns quickly in the first 1250 episodes, after which it stabilises around a reward value of -25. This means that on average the pursuer manages to catch the target in 35 steps. The results stay rather spread out and in late episode the pursuer still does not manage to always catch the target. In figure 4 the pursuer needs to catch a target that acts completely random. Against random behaviour the pursuer manages to catch the target more often early on, however it takes the pursuer more episodes to stabilise. Eventually the pursuer will stabilise around a value of -25. In the appendix there are variations on the variables used for the first figure. When lowering α or γ the pursuer seems to learn at the same rate and stabilise around the same value. Lowering ε to 0.75 seems to increase the early learning rate of the pursuer, though apart from that it behaves similarly to the first values. Changing ε to a lower value seems to have no noticeable impact. In the appendix there is also a figure where the values of α, γ and ε did not decay overtime, this resulted in a pursuer that was not able to improve beyond its initial ability.

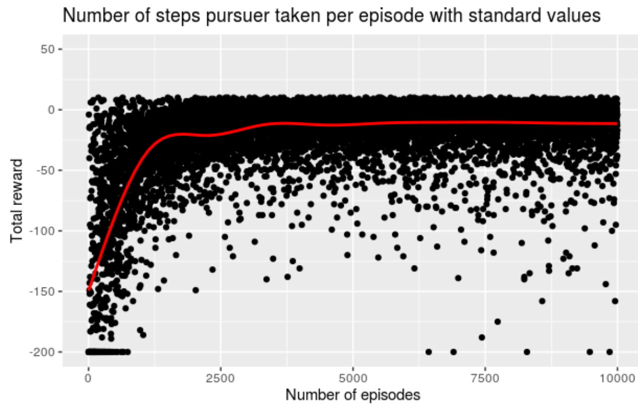

Figure 3: The reward earned by the pursuer per episode

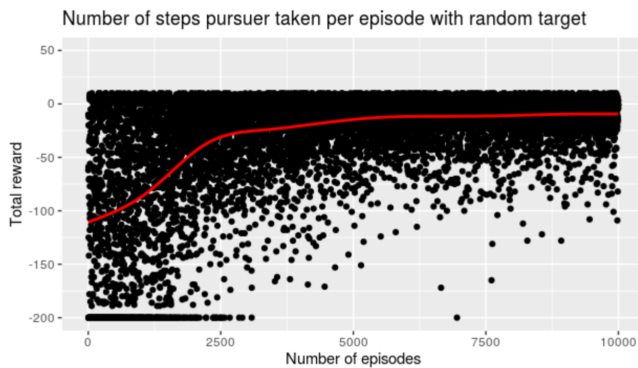

Figure 4: The reward earned by the pursuer per episode against a random acting target

5. Conclusion

This research shows that in a grid environment a pursuer can learn to catch an evader using Sarsa learning algorithm. This is both the case for an evader that moves away from the pursuer and an evader that moves randomly. It also showed that, for a 10x10 grid, 200 steps are needed per episode in order for the pursuer to learn quickly. Changes of around 0.2 in the variables α, γ and ε seem to have little impact on the learning ability of the pursuer. Though in none of the circumstance did the pursuer always catch the evader. The spreading in of the rewards the pursuer got per episode is probably varied so much due to the random starting locations of both agents, this means that the minimum number of steps required to even reach the evader can vary a lot.

The biggest limitation of this research is the limited amount of time available. It could have been more insightful to have the pursuer train against a more complex evader that might use wall more effectively. A simulation of the game might give more insight in what decisions the pursuer makes and why it sometimes does not manage to catch the evader in the later episodes.

There is some potential for Sarsa to work in pursuit-evasion games, though it will have to be tested and trained in more complex scenarios to see if it can keep up with the complexity of the environment. However Sarsa could be used for simpler applications like in games, however the exact applications for Sarsa in pursuit-evasion algorithms is a matter of future research.

6. References

1. Akhloufi, M. A., Arola, S., & Bonnet, A. (2019). Drones Chasing Drones: Reinforcement Learning and Deep Search Area Proposal. *Drones*, *3*(3), 58. https://doi.org/10.3390/drones3030058
2. Chung, T. H., Hollinger, G. A., & Isler, V. (2011). Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, *31*(4), 299–316. https://doi.org/10.1007/s10514-011-9241-4
3. Desouky, S. F., & Schwartz, H. M. (2011). Q(λ)-learning adaptive fuzzy logic controllers for pursuit-evasion differential games. *International Journal of Adaptive Control and Signal Processing*, *25*(10), 910–927. https://doi.org/10.1002/acs.1249 https://doi.org/10.1145/2816839.2816925
4. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series)* (second edition). Consulted from https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

7. Appendix

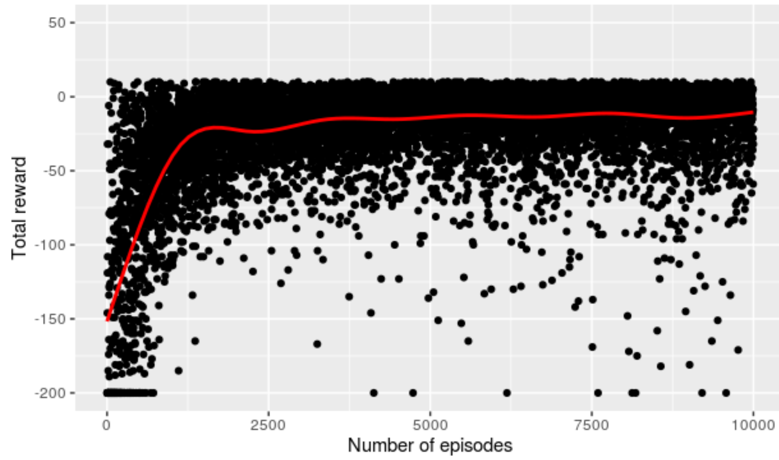

Figure 3

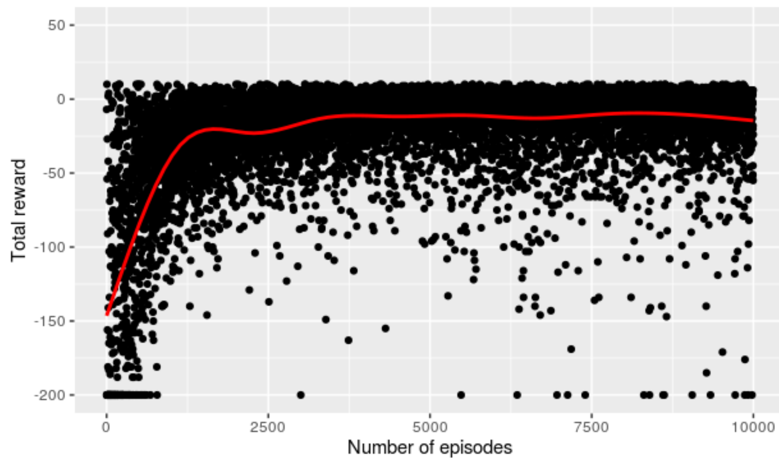

Figure 6

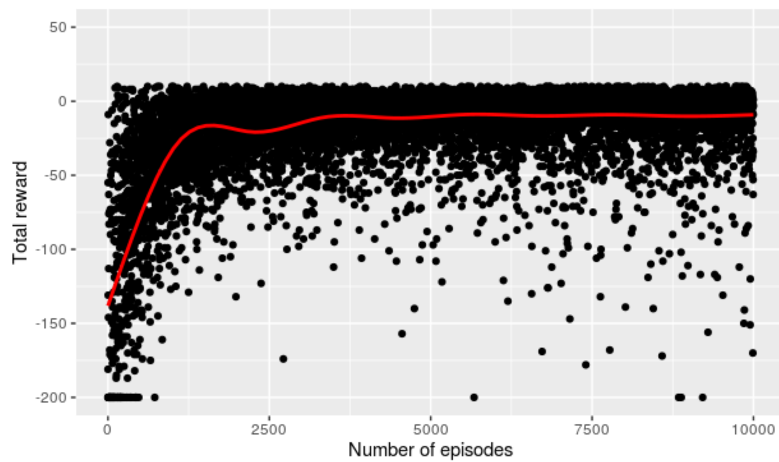

Figure 7

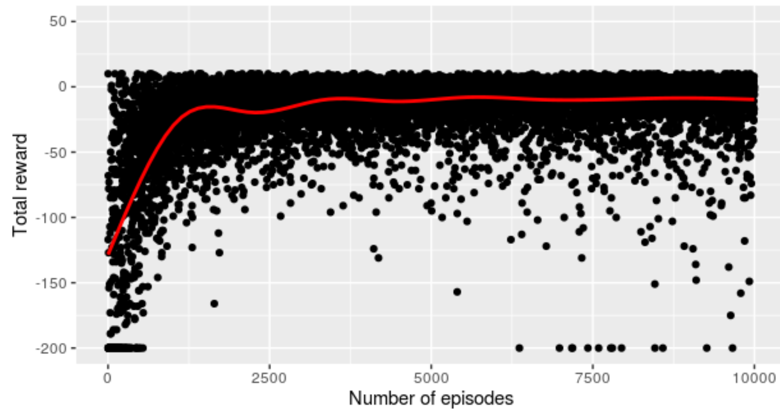

Figure 8

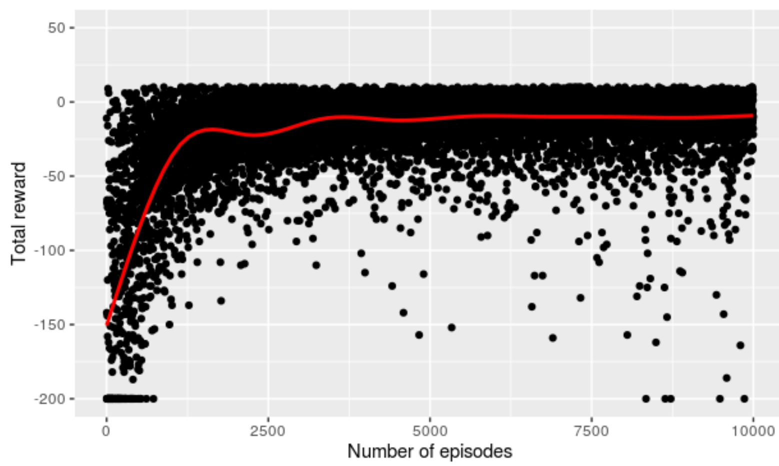

Figure 9

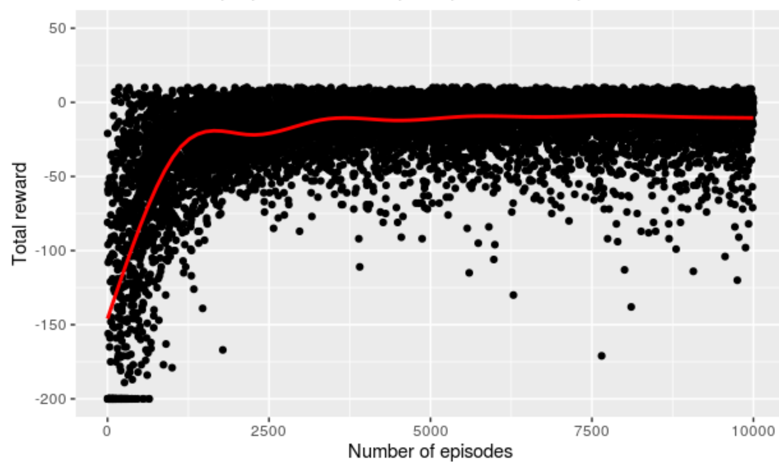

Figure 10

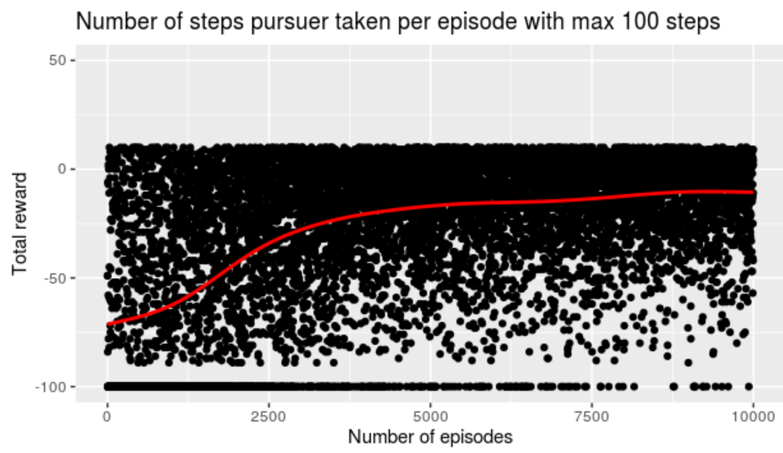

Figure 11

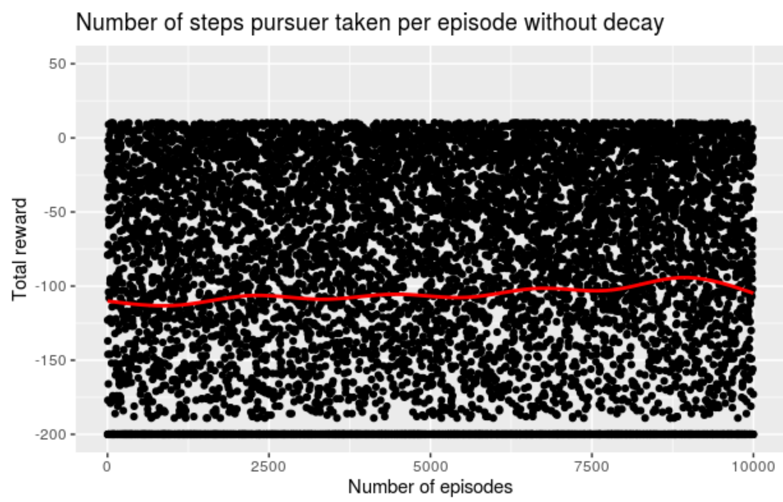

Figure 12