UNIVERSITATEA DIN
BUCUREȘTI

FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Disertație

REINFORCEMENT LEARNING IN VIDEO GAMES

Absolvent
Smarandache Mihnea

Coordonator științific
Păduraru Ciprian Ionuț

București, septembrie 2023

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce vitae eros sit amet sem ornare varius. Duis eget felis eget risus posuere luctus. Integer odio metus, eleifend at nunc vitae, rutrum fermentum leo. Quisque rutrum vitae risus nec porta. Nunc eu orci euismod, ornare risus at, accumsan augue. Ut tincidunt pharetra convallis. Maecenas ut pretium ex. Morbi tellus dui, viverra quis augue at, tincidunt hendrerit orci. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam quis sollicitudin nunc. Sed sollicitudin purus dapibus mi fringilla, nec tincidunt nunc eleifend. Nam ut molestie erat. Integer eros dolor, viverra quis massa at, auctor.

Contents

Chapter 1

Introduction

TODO: Ceva despre jocuri video, ceva despre AI e important, ceva ca cu cat avanseaza jocurile devin mai complexe si e mai greu sa faci un AI whatever cv de genu (2 pag max)

Chapter 2

Preliminarii

TODO: aici vreo 3-4 pagini max nu vreau sa scriu prea mult dar tre sa ma bazez pe referinte

2.1 PPO

2.2 Unity

2.3 ML-Agents Toolkit

2.4 Testing game TODO: change the title

Chapter 3

Agent Implementations

3.1 Reaching a static target

3.1.1 The Problem

A common task for an AI in a game is to reach a a given destination. According to [1] the common way to achieve this goal is by using *navigation meshes* (NavMeshes). These are represented as graphs, and their nodes represent surfaces that can be traversed. Afterwards, algorithms such as $A*$ can be used to find the fastest way for an agent to go from point A to point B. However, these *NavMeshes* require to be *baked* (TODO explicatie baked) in advance, so updating them in real time can prove to be a challenge depending on several factors such as:

- the game engine used
- the complexity of the game environment
- the computational cost associated with recreating in real time these meshes

A proposed solution for this problem would be to use AI agents that have been trained using deep learning methods, in such a manner that they would be independent from changes to the environment.

3.1.2 Implementing the solution

First, the action space for the agent is defined: due to the fact that we only need to move the agent to a given position, the action space consists of moving the agent forward or backward, and rotating it. Because the agent is supposed to be controlled by a controller, its movement input is defined as a real number in the $[-1, 1]$ interval for both X and Y axes. However, a discrete action space will be used instead of a continuous one to reduce the difficulty of learning, and also because there is no need for the agent to have

movements that are so precise. For each axis the action space will be represented by the discrete space: $\{-1, -0.5, 0, 0.5, 1\}$.

To make decisions, the agent will need to make several observations about its surroundings. Firstly, it will need to know how close it is to certain objects in the environment. To accomplish this, several *raycasts* will be used to measure the distance from the agent, similar to a *LIDAR* (TODO: add ceva despre lidar si un paper). The raycasts start from the center of the agent and are spread in such a way that the angle between 2 consecutive rays is equal for any 2 consecutive rays (TODO: imagine cu rays, si probabil sa gasesc un termen mai ok). The observation will contain the distance until the rays hit an object. Also, the index of the layer of the hit object will be included in the observations, so that the agent will differentiate between regular environment objects and more important objects, such as other players, enemies, etc. For this implementation, 32 rays were used.

Other observations that are made are the agent's position in space and also that of the target. This observation is included so that the agent can learn how movement brings it closer or further to the target. The next observation is the agent's forward vector so it can know in which direction it is moving. The optimal direction that the agent should take is also observed and obtained by computing the vector difference between the target's position and the agent's position. To know how much to adjust its trajectory, the angle between the agent's forward vector and the target is observed; the angle is signed so that the agent can learn to adjust its trajectory to the left or to the right. The distance to the target is added to the observations so that the agent can learn that when the distance is getting smaller it is rewarded. The agent's normalized velocity vector is observed to show in which direction it is moving based on the given input. The angle between the agent's velocity vector and its forward vector is observed to tell if the agent is moving forwards or backwards. Finally, the agent's velocity magnitude is observed so the agent can know if it is moving or standing still.

In summary the following observations are being made:

- distance for each raycast until it hits an object
- layer index of object hit by the raycast
- spatial postion of the agent
- spatial position of the target
- agent's forward vector
- optimal direction of the agent
- signed angle between the agent's forward vector and the target

- distance from the target
- agent's normalized velocity vector
- angle between the agent's velocity vector and its forward vector
- agent's velocity magnitude

In order for the agent to be able to learn to solve a specific problem, in this case, to reach a target, it must be rewarded or punished according to the actions that were taken. It is important to be mindful of the rewards that are given to the agent because only giving rewards and not punishing it can lead the agent to learn a behaviour that will maximize its reward, but will not be able to solve the problem, as it will be described shortly.

To begin, the first reward that was implemented, was the reward for reaching the objective, which is the goal of the agent and should also be a substantial reward. The other rewards that ware added were based on the direction of the movement and how close to the target it was, the reward increasing in value if the agent was closer to the objective (3.1) and a reward if the raycasts are hitting the destination object, which also increases in value if the agent is closer to the target (3.2).

$$R = \frac{(1 - \frac{\alpha}{180}) \cdot r}{d} \tag{3.1}$$

where:

R : is the total reward

α : is the angle between the agent's current direction and the optimal direction

r : is the reward that is obtained if the agent has the optimal direction

d : is the distance from the agent to the objective

$$R = \frac{r}{d_i} \tag{3.2}$$

where:

R : is the total reward

r : is the reward obtained if the distance between the agent and the objective is minimum

d_i : is the obtained distance by the raycast i from the agent to the object

However, these three rewards are not enough for the agent: through learning experiments it was observed that the agent was performing poorly: it would get stuck trying to move through a wall, it would never reach the objective and just spin around, or in some cases, it would just stand still.

To solve these problems, several punishments were implemented to correct the behavior of the agent. To prevent the agent for not moving, a constant penalty was added, to incentivise the agent to move towards the reward, so that it will receive a reward. Also, to comabat standing still, if the agent does not move, it receives an additional penalty, and if it does not move for more than 100 time steps, it receives a huge penalty and the episode ends.

To stop the agent from trying to pass thorugh walls, a penalty is added if the raycasts that hit walls or other objects in the environment, have the distance to the hit object be a smaller than a given number. This penalty also increases the closer the agent gets to a wall (3.3).

$$R = \frac{p}{d_i}, \text{if } d_i < d \tag{3.3}$$

where:

R : is the total reward

p : is the penalty obtained if the distance between the agent and the wall/environmental object is minimum

d_i : is the obtained distance by the raycast i from the agent to the object

d : is the maximum distance for which the penalty is applied

Another penalty was added if the agent is moving away from objective, and as before, it increases the further away it gets from the objective (3.4).

$$R = \frac{\alpha}{180} \cdot p \tag{3.4}$$

where:

R : is the total reward

p : is the penalty obtained if the distance between the agent and the wall/environmental object is minimum

α : is the angle between the agent's current direction and the optimal direction

Through training, two undesirable behaviours were observed: it was observed that the agent would sometimes make sudden jerky movments, trying to change its direction and

Name	Value	Notes
Reach Objective Reward	10	
Move Towards Objective Reward	0.001	is scaled by the distance between agent and objective
Raycast Touches Objective Reward	0.001	is scaled by the distance between agent and objective
Constant Penalty	-0.005	is applied at each time step
Not Moving Penalty	-0.05	
Not Moving For 100 Steps Penalty	-50	
Moving Towards Wall Penalty	-0.002	is scaled by the distance between agent and object
Moving Away From Objective Penalty	-0.025	is scaled by the distance between agent and objective
Sudden Movement Penalty	-0.01	is scaled by the angle between the agent's current direction and its previous one
Moving Backwards Penalty	-0.005	

Table 3.1: Rewards and Penalites

immediatly returning to its previous trajectory, and that the agent would learn to drive backwards. To fix the first problem, a penalty was added if the agent would change its direction (3.5), and to fix the second one, a penalty was added if the tank was moving backwards (3.6).

$$R = \frac{\alpha}{180} \cdot p \tag{3.5}$$

where:

R : is the total reward

p : is the penalty obtained for changing the movement direction

α : is the angle between the agent's current direction and its previous direction

$$R = \frac{\alpha}{180} \cdot p, \text{if } \alpha > 90 \tag{3.6}$$

where:

R : is the total reward

p : is the penalty obtained for changing the movement direction

α : is the angle between the agent's current direction and its forward vector

In summary, the used rewards and penalites and their values can be seen in table 3.1

Hyperparameter	Value
Gamma	0.9
Lambda	0.95
Beta	0.005
Epsilon	0.2
Buffer Size	8192
Batch Size	256
Number of Epochs	3
Learning Rate	0.0003

Table 3.2: Training hyperparameters

3.1.3 Training

Each training session consisted of 10^7 steps, and 30 agents were trained in parallel. The training times were between 3-4 hours. The agents are supposed to learn to reach an objective which appears in one of 17 predefined positions on the map. Once the agent reaches the objective, it is moved in another location chosen randomly. Each separate training instance uses the same seed for the random number generator, so that the objectives will appear in the same order for each of the training instances. Each training episode has a limit of 5000 steps.

In the initial training sessions, the agents were unable to learn to reach the objective, becoming stuck rotating in a circle. A proposed solution was to keep the objective in the same position until the agent reaches it 30 times. After reaching the objective 30 times, the objective would start appearing in the random predefined locations. This was done to possibly kickstart the agent's learning process, but the approach failed, the agent being unable to learn to reach the objective. Another approach was to increase the neural network's size, however this approach prove unsuccessful as well. The approach that worked was to remove the agent's postion and the objective's position from the observations. TODO: sa zic de curiosity

The agent's training process was done using multiple neural network configurations, including different number of network layers (1, 3, 5, 7) and different number of units per layer (128, 256, 512).

The hyperparameters used for the training were:

The results for training the agent with 1 network layer can be seen in Figure 3.1, and in Table 3.3. In this case, the configuration with 256 units was not able to be trained properly, remaining stuck moving in a circle. Between the two configurations that were successfully trained, it can be seen that the one with 128 units, performed better during the training process than the one with 512 units, obtaining, at the end, a mean reward that is higher by 18.79%.

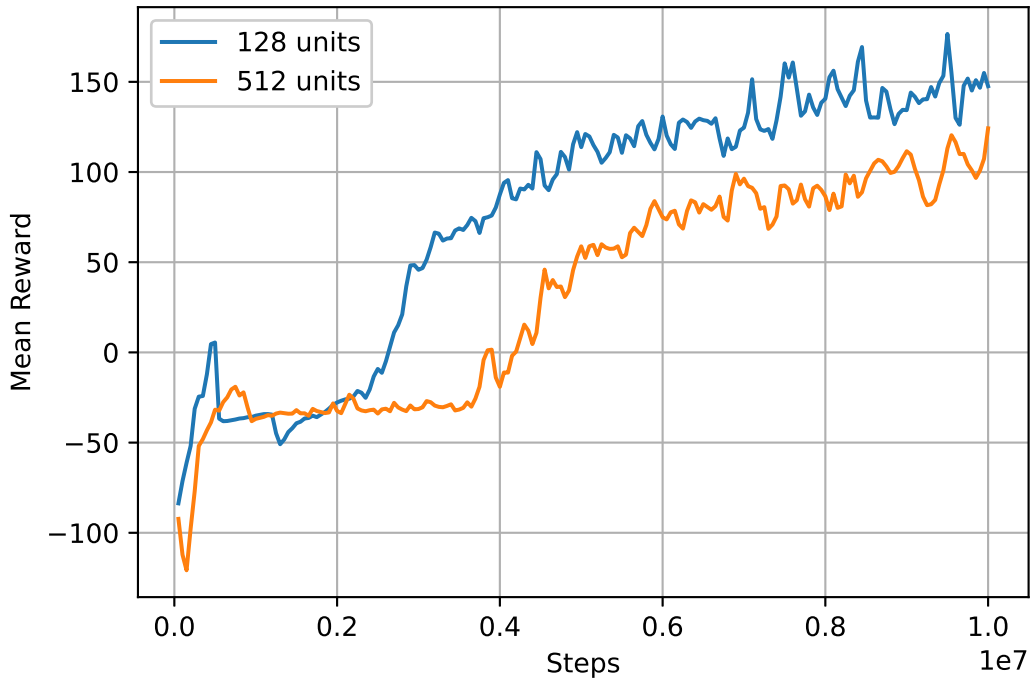

Figure 3.1: Training results for reaching a static target with a network with 1 hidden layer

The results for training the agent with 3 network layers can be seen in Figure 3.2, and in Table 3.3. From these results, it can be seen that the configurations with 128 and 256 units, performed very similarily during the training, however, the one with 512 units started to lag behind these two at around $4 \cdot 10^6$ steps. Again, the configuration with the least units had the best training results, obtaining a result that is 2.61% better than the one with with 256 units and better by 29% than the one with 512 units.

Results for training the agent with 5 network layers can be seen in Figure 3.3, and in Table 3.3. From the training results, it can be seen that in the later stages, the configuration with 256 units falls behind the other two. Unlike the results with 1 layer and 3 layers, the best performing configuration is the one with 512 units per layer, being better by 8.72% than the configuration with 128 units, and by 35.45% than the configuration with 256 units.

Results for training the agent with 7 network layers can be seen in Figure 3.4, and in Table 3.3. From the training results it can be seen that the configuration with 128 units per layer performs better than the other two, but at the final stages of the training, it is caught up to by the configuration with 256 units. In the end, the configurations with 128 and 256 units per layer had virtually identical results at the end of the training, and were better by $\sim 36.3\%$ than the one with 512 units.

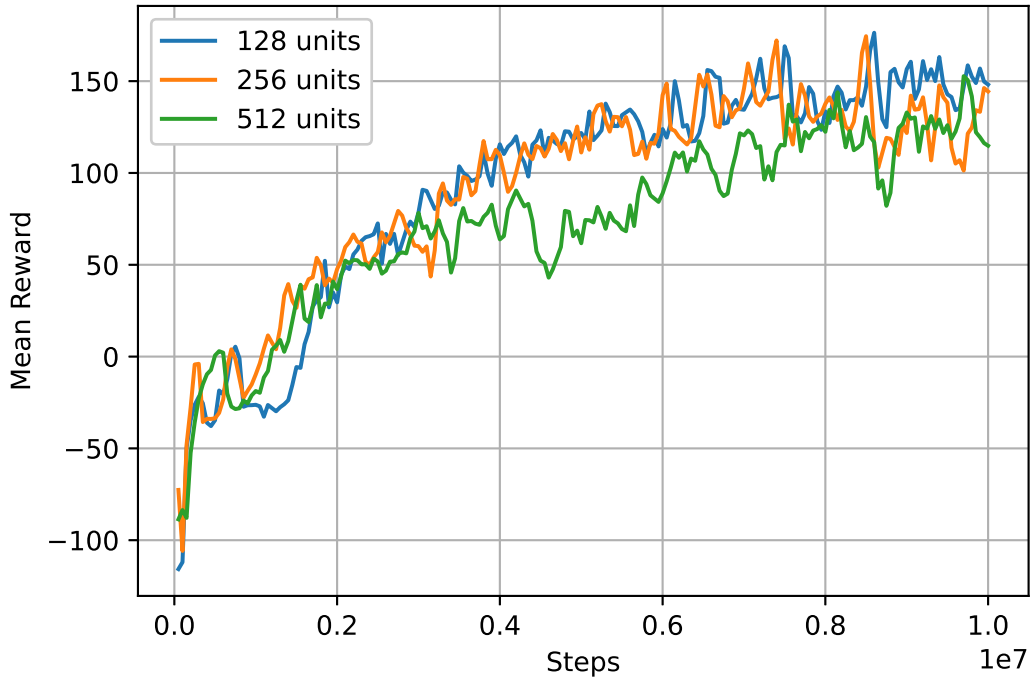

Figure 3.2: Training results for reaching a static target with a network with 3 hidden layers

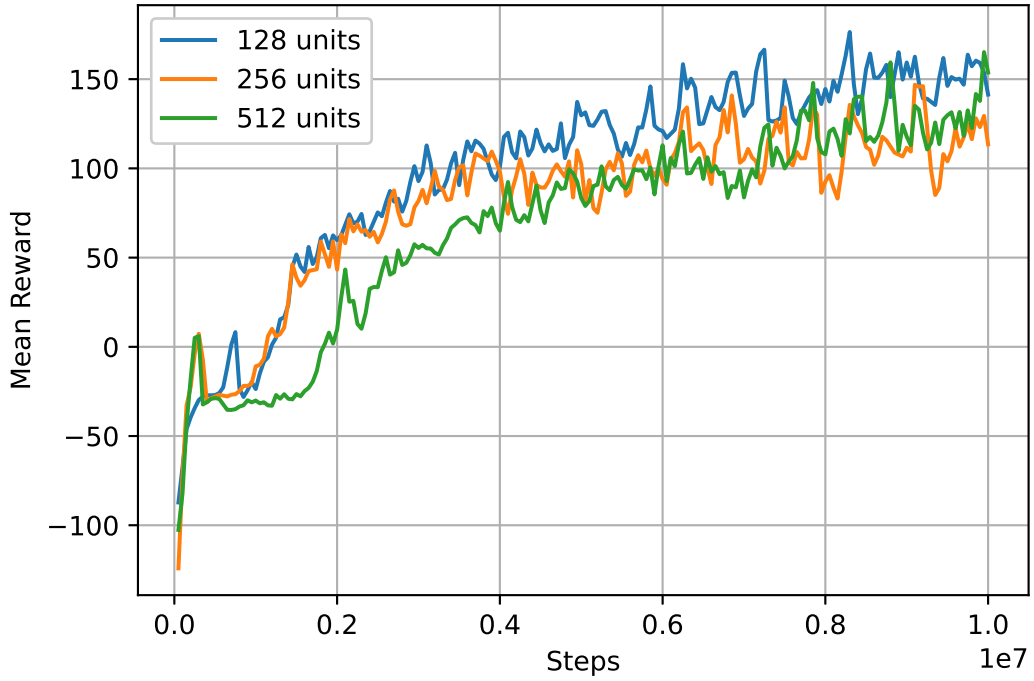

Figure 3.3: Training results for reaching a static target with a network with 5 hidden layers

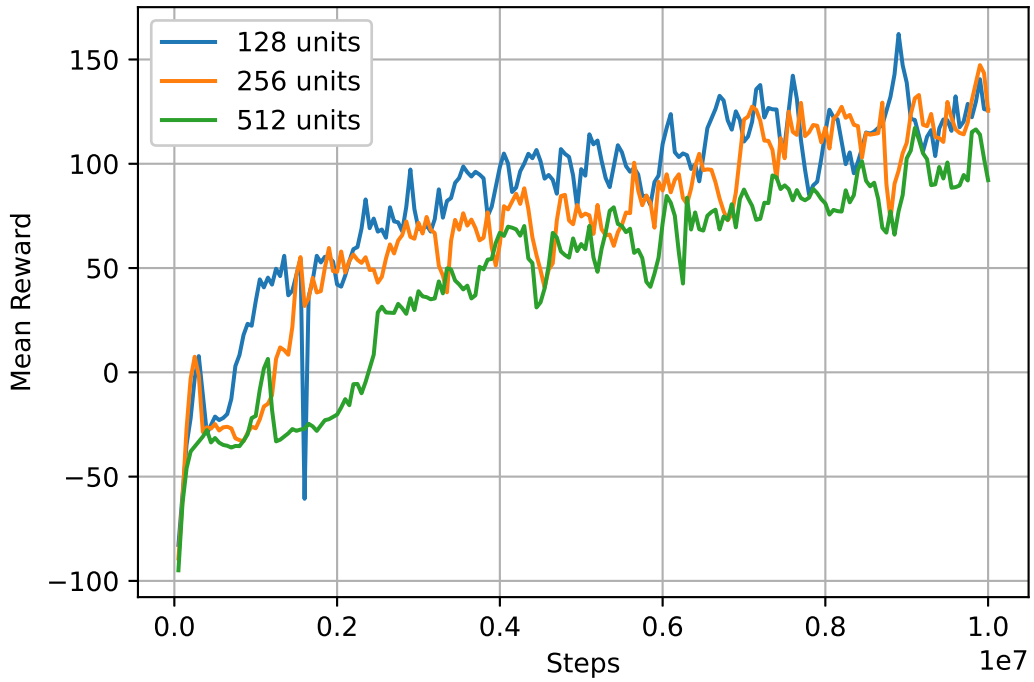

Figure 3.4: Training results for reaching a static target with a network with 7 hidden layers

All the training results can be seen in Table 3.3 and in Figure 3.5. It can be observed that network configurations with a smaller number of units per layer obtained better training results than the others. This could be due to the fact that the problem is not a very complex one, and because a smaller network can be trained faster. The configurations with 1, 3, or 5 layers obtained similar results, however, when 7 layers are used, there is a noticeable decrease in the agent's performance. These being said, from the training results it seems that using a network configuration with less layers and less units per layer offers almost the best result, thus it is the most practical one to be used, since it uses the least computing resources. Using larger networks should be done by training the agent for longer periods of time. However, it is possible when using a smaller network to obtain an unusable model, as was the case in this experiment using a network configuration with 1 hidden layer and 256 units per layer.

3.2 Reaching a moving target

3.2.1 The Problem

Another standard behavior of an AI in videogames is to be able to chase a target, so that it can, maybe, perform a specific action once the object controlled by the AI is close

Network Configuration	Final Mean Reward
1 layer, 128 units	147.568
1 layer, 256 units	DNF
1 layer, 512 units	124.221
3 layers, 128 units	148.136
3 layers, 256 units	144.366
3 layers, 512 units	114.828
5 layers, 128 units	141.369
5 layers, 256 units	113.478
5 layers, 512 units	153.709
7 layers, 128 units	125.727
7 layers, 256 units	125.441
7 layers, 512 units	92.128

Table 3.3: Final training results for reaching a static target

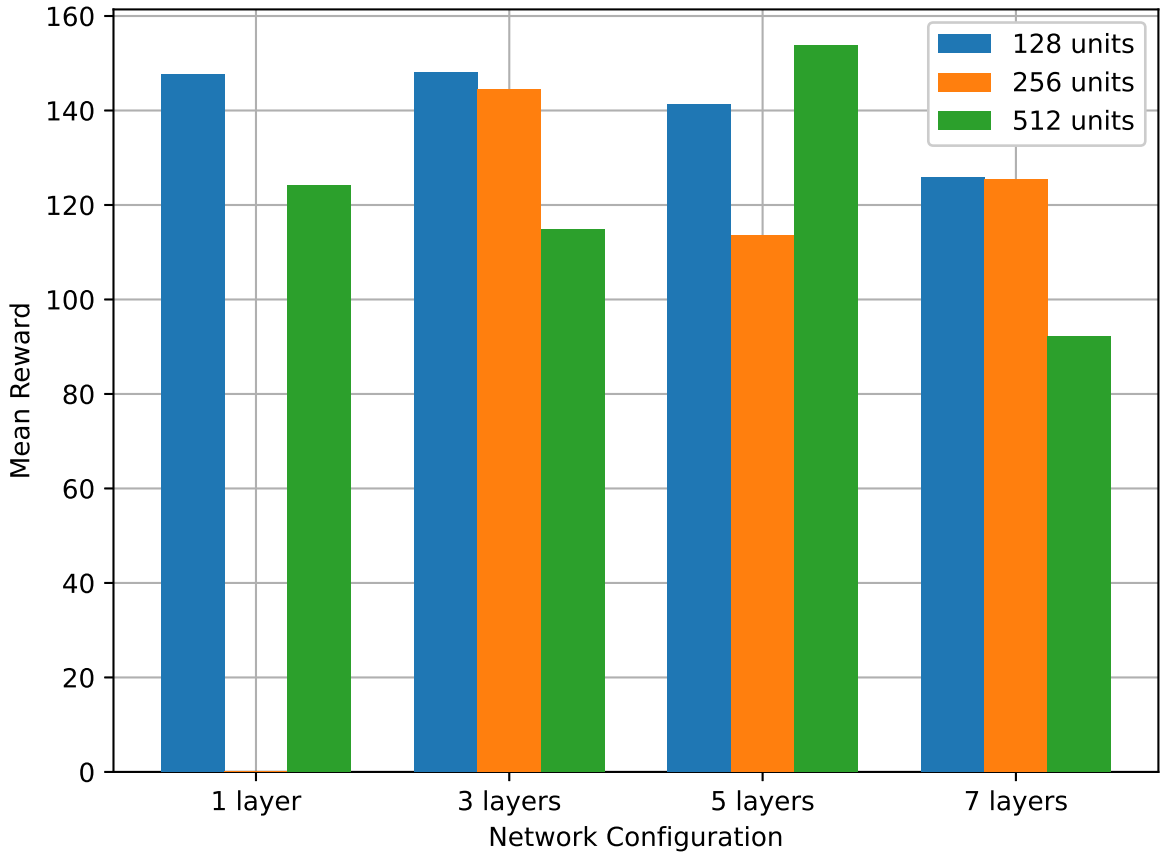

Figure 3.5: Training results for all network configurations for reaching a static target

enough to the target (TODO: sa mai dezvolt asta). A simple way to achieve this behavior would be to use a *NavMesh* and periodically change the agent's destination. Compared to the problem described at 3.1.1, the addition here is that the agent has to recompute its route at specified intervals.

Again, the proposed solution would be to use AI agents that are trained using deep learning methods.

3.2.2 Implementing the solution

The basic implementation for this problem is the same as the one described at 3.1.2, with the same observations, rewards and penalties being used. The addition would be to add an observation that is the direction in which the target is moving. This can be used by the agent to predict in which direction the target is moving and to eventually intercept it along its trajectory. Another idea would be to add recurrent memory (TODO: sa zic ce e recurrent memory) to the agent so that it can achieve this result in a more "natural" way.

3.2.3 Training

The training is similar to the process described at 3.1.3, with the same number of agents being trained at the same time, same number of steps, same hyperparameters (Table 3.2), etc.

Trying to add recurrent memory to the agent was a failure, the agent being unable to learn to chase the target, and getting stuck moving in a circle. This behaviour happened even when changing the recurrent network's hyperparameters *sequence length* and *memory size*. (TODO: maybe add some combinations)

Training this agent was done in 2 separate ways: one where the target's direction is not in the observations, and one where it is. The first round of training is done without the observation that was mentioned previously.

Results for training the agent with a single network layer and without the target's direction obsrevation can be seen in Figure 3.6 and Table 3.4. From these results it can be seen that all 3 configurations performed virtually identically, and at the end of the training had obtained the same mean reward.

Results for training the agent with 3 network layers and without the target's direction obsrevation can be seen in Figure 3.7 and Table 3.4. In these results, it seems that the configuration with 128 units obtains better results during training than teh configuration with 256 units, which in turn obtains better results than the one with 512 units. At the

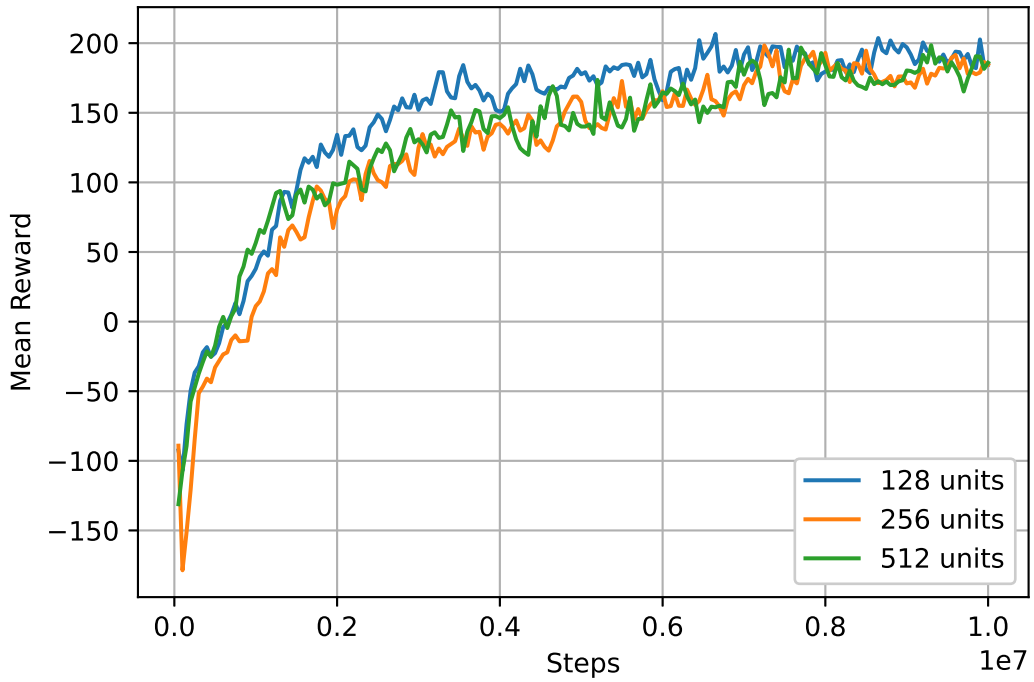

Figure 3.6: Training results for reaching a moving target with a network with 1 hidden layer

end of the training, the configuration with 128 units is better by 1.02% than the one with 256 units and better by 28.58% than the one with 512 units.

Results for training the agent with 5 network layers and without the target's direction obsrevation can be seen in Figure 3.8 and Table 3.4. The results during training are similar to the configuration with 3 layers, with the configuration with 128 units being better than the other two. The final result is that the configuration with 128 units is better by 8.87% than the one with 256 units and better by 17.9% than the one with 512 units.

The second round of training is done by adding the target's movement direction to the agent's observations.

The results for training the agent with a single network layer and without the target's direction obsrevation can be seen in Figure 3.9 and Table 3.4. As in the previous cases, the best performed configuration during training is the one with the least units, i.e. 128 units. This one is better by 39.47% than the configuration with 256 units, and by 6.88% than the one with 512 units.

Training the agent with a network with 3 layers obtains the results that can be seen in Figure 3.10 and Table 3.4. This time, all 3 configuration performed very similarily during training. The best configuration here was the one with 512 units which was better by

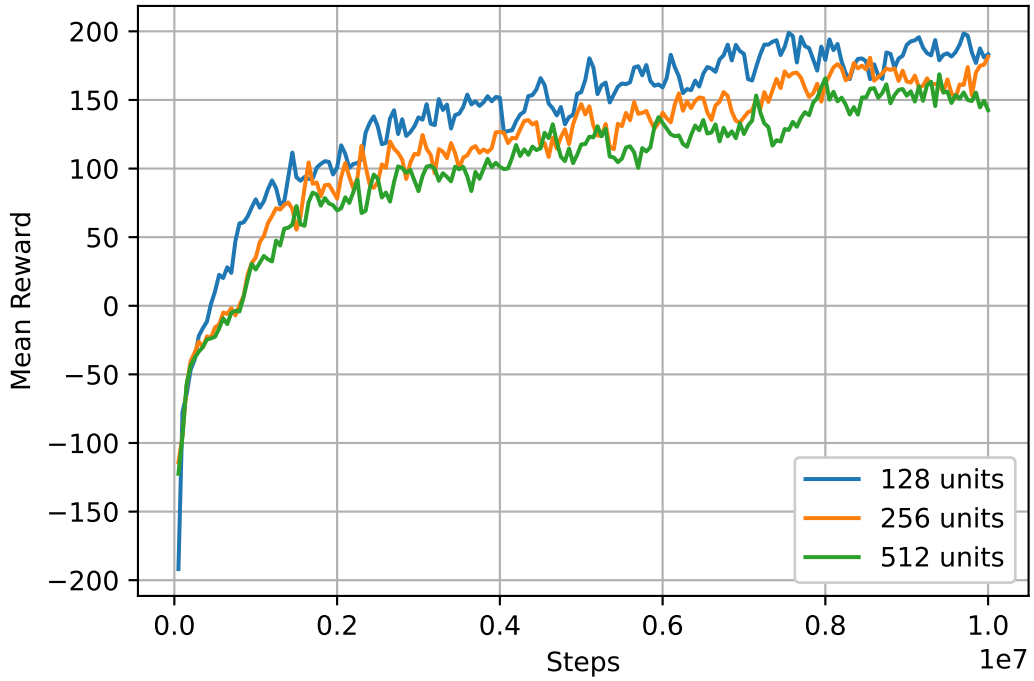

Figure 3.7: Training results for reaching a moving target with a network with 3 hidden layers

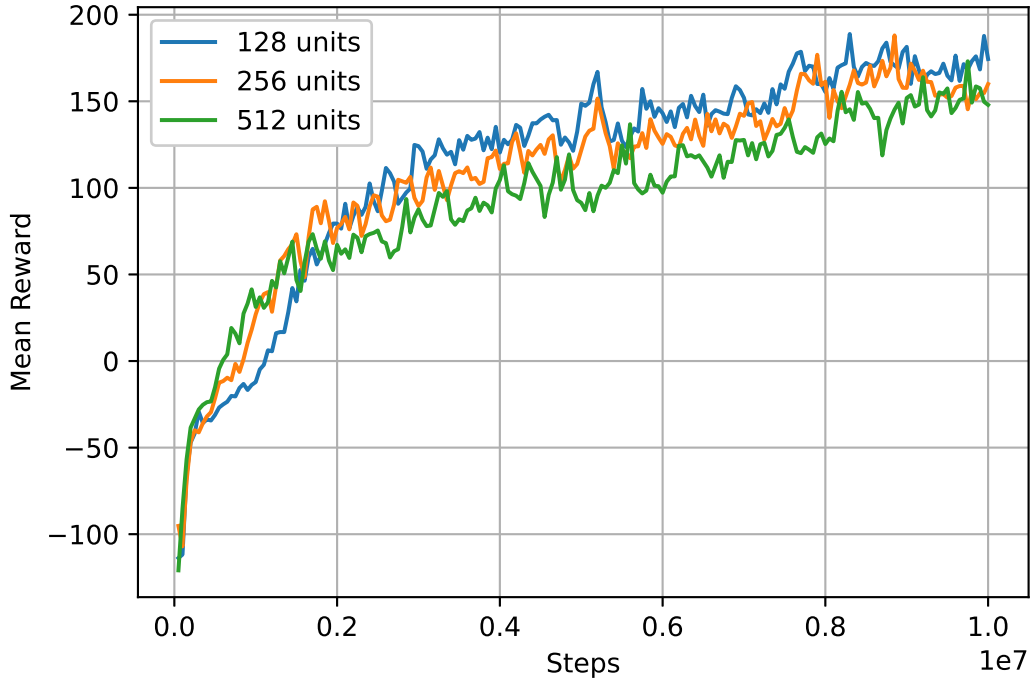

Figure 3.8: Training results for reaching a moving target with a network with 5 hidden layers

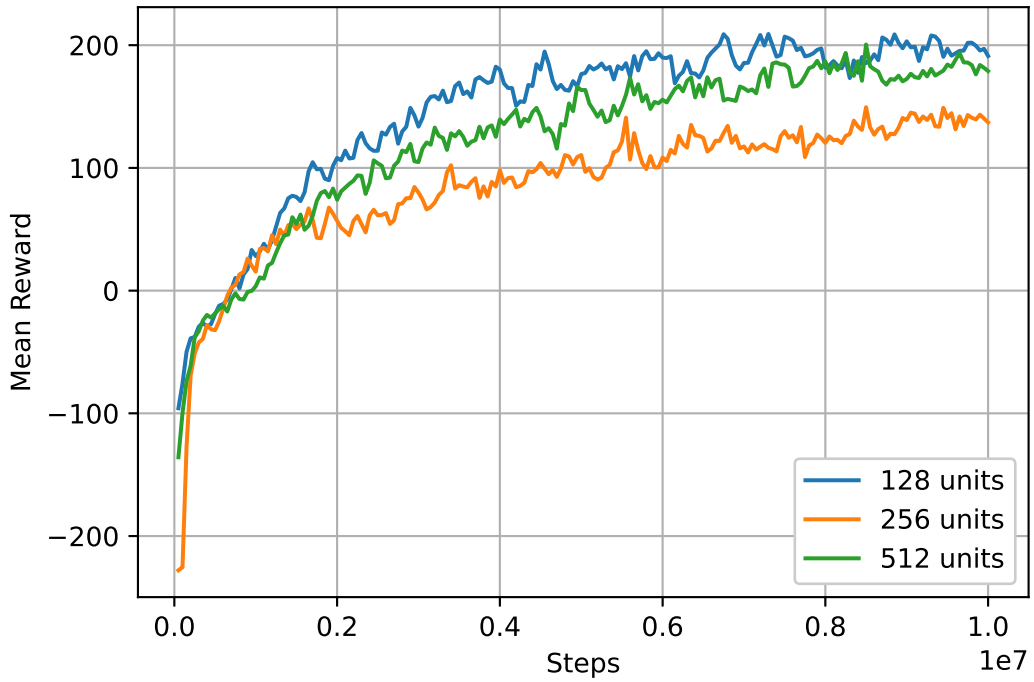

Figure 3.9: Training results for reaching a moving target with a network with 1 hidden layer and with observation of target's direction

4.99% than the one with 128 units, and by 7.37% than the one with 256 units.

Results for training the agent with 5 network layers and without the target's direction obsrevation can be seen in Figure 3.11 and Table 3.4. It can be observed that the configuration with 128 units outperforms the other two during training, the ohter two performing mostly identically until the final training steps, where the one with 512 units slightly overtakes the one with 128 units. The configuration with 512 units is better by 1.26% than the one with 128 units and better by 17.93% than the one with 256 units.

Looking at the final results in Figure 3.12 and Table 3.4 we can observe that the best netowrk configurations have either 128 units per layer or 512 units per layer. When increasing the number of layers and not using the target's movement direction as an observation, the agent's performance slightly decreased during the training. Adding the target's movement direction as an observation increases the performance of the agent, compared to the configuration with the same number of layers and without this observation.

In conclusion, from these training results, it seems that adding the target's moving direction to the observations improves the agent's performance, the best results are achieved with either 128 or 512 units per hidden layer, and the best number of hidden layers is 3. The difference between using 128 units instead of 512 is a $\sim$ 5% decline in the agent's

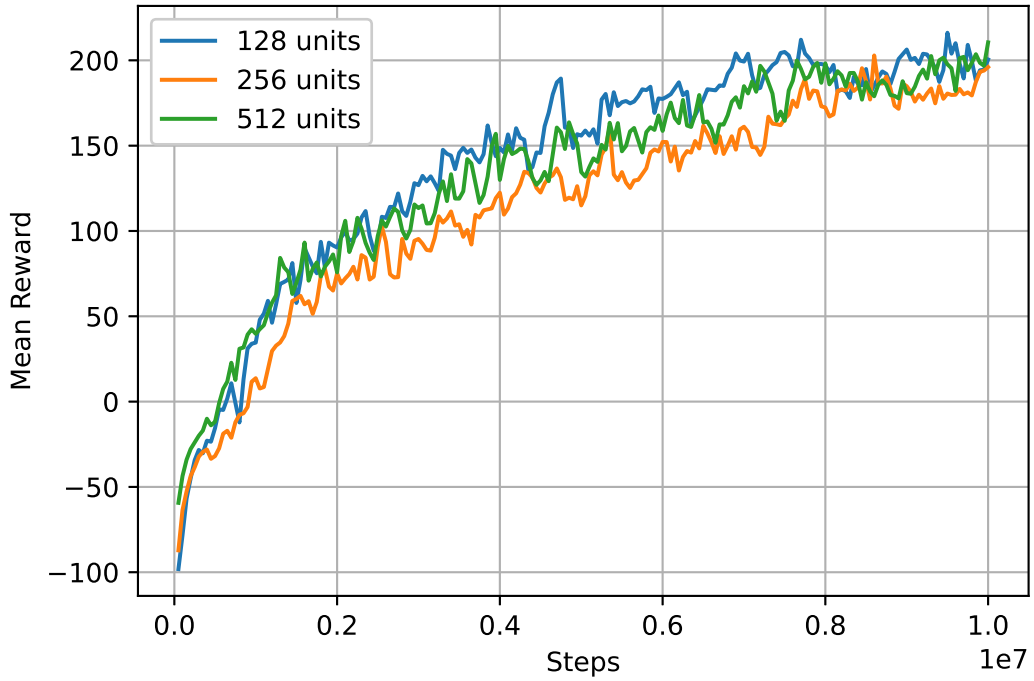

Figure 3.10: Training results for reaching a moving target with a network with 3 hidden layers and with observation of target's direction

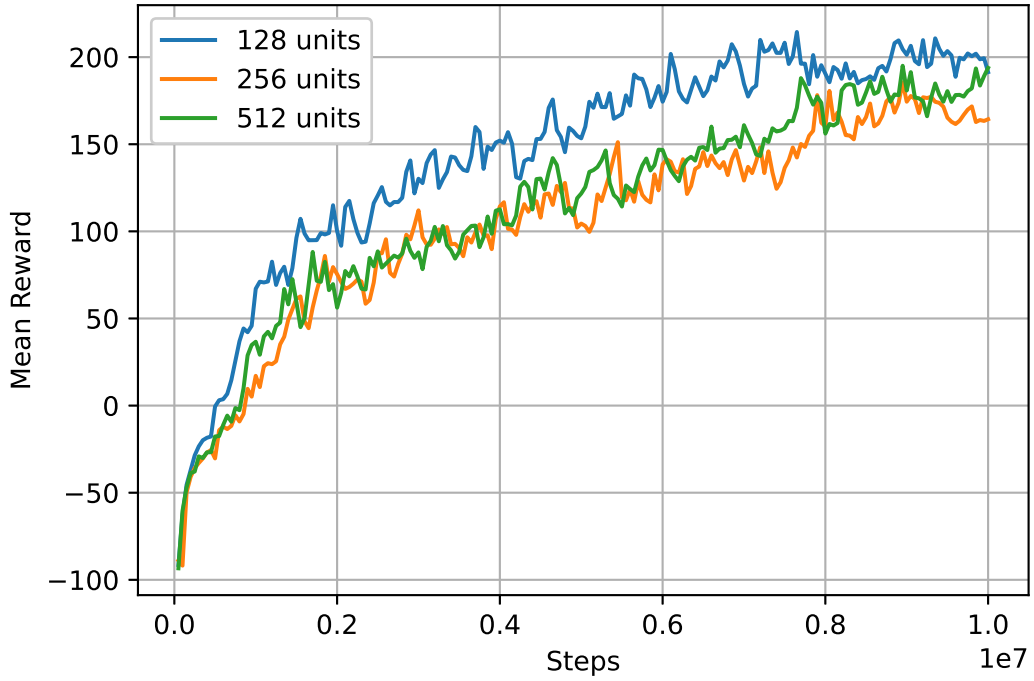

Figure 3.11: Training results for reaching a moving target with a network with 5 hidden layers and with observation of target's direction

Network Configuration	Observed target's direction	Final Mean Reward
1 layer, 128 units	No	184.547
1 layer, 128 units	Yes	191.184
1 layer, 256 units	No	184.563
1 layer, 256 units	Yes	137.075
1 layer, 512 units	No	185.936
1 layer, 512 units	Yes	178.876
3 layers, 128 units	No	183.288
3 layers, 128 units	Yes	200.502
3 layers, 256 units	No	181.435
3 layers, 256 units	Yes	196.064
3 layers, 512 units	No	142.54
3 layers, 512 units	Yes	210.524
5 layers, 128 units	No	174.404
5 layers, 128 units	Yes	191.399
5 layers, 256 units	No	159.966
5 layers, 256 units	Yes	164.351
5 layers, 512 units	No	147.922
5 layers, 512 units	Yes	193.828

Table 3.4: Final training results for reaching a moving target

performance. If there are no reasons to reduce resource usage, the configuration with 3 hidden layers and 512 units per layer should be used.

3.3 Shooting a moving target

3.3.1 The Problem

Another common use of AIs in gaming is to provide an opponent for the player, in this case to fight against. This brings a whole new dimension to the problem besides having the AI move in the given worldspace. (TODO: more research pe cacatu asta)

3.3.2 Implementing the solution

The implementation builds on what was described in Sections 3.1.2 and 3.2.2. The addition is that the agent does not need to reach a certain target, but to shoot it. For this, several aspects need to be changed to implement the bullet logic.

For starters, the agent's action space was increased from 2 branches to 4 branches. The first 2 branches will still handle the agent's movement, the third branch can have two values: 0 or 1, and this tells the agent if he should fire a bullet, and the fourth branch

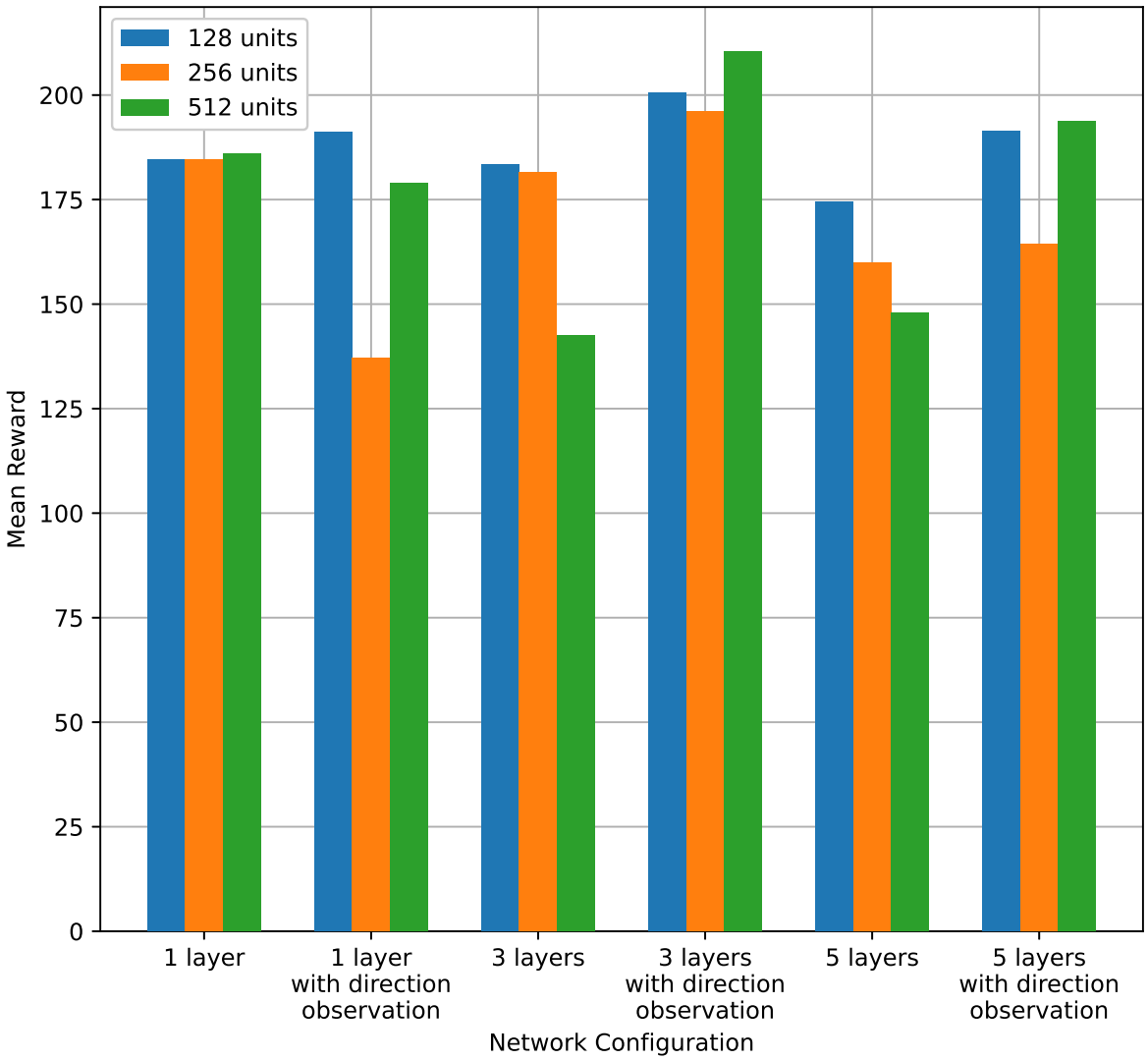

Figure 3.12: Training results for all network configurations for reaching a moving target

sets the bullet's shooting force, which can have 3 values: low, medium, or high.

Another change that was made, was adding new observations, most of them being related to the bullet. The first one is a flag that tells if the bullet is fired. The second one is the bullet's trajectory, which can be used by the agent to learn to shoot on target. The third observation is about the bullet's speed so that the agent can learn with what force the bullet has been shot. The final new observation, which is not related to the bullet, is the angle between the target's front vector and the vector from the target to the agent. This is used so that the agent can learn when it is in the target's crosshairs, and to possibly learn how to flank the target.

In summary, the following observations have been added:

- bullet is fired or not
- bullet's trajectory vector
- bullet's speed
- angle between the target's front vector and vector from target to the agent

Finally, new rewards and punishments were implemented. Firstly, the agent is no longer rewarded if it touches the target, it has to hit it with a bullet. To incentivise firing a bullet, a small reward is received when firing a bullet. Also, a penalty is added if the bullet misses the target, and disappears because it has travelled a certain distance, or it hit an environmental object. The other bullet reward that was added is a reward that is based on the bullet's trajectory and how close it is to the optimal trajectory to the target and is scaled by the distance from the bullet to the target, with the reward increasing the closer the bullet gets to the target. This reward is defined using Formula 3.7:

$$R = \frac{(1 - \frac{\alpha}{180}) \cdot r}{d} \cdot 2 \tag{3.7}$$

where:

R : is the total reward

α : is the angle between the bullet's current trajectory and the optimal trajectory

r : is the reward that is obtained if the bullet has the optimal trajectory

d : is the distance from the bullet to the target

Two punishments were added to help the agent learn as if it were fighting a real opponent. The first one makes the agent stay at a given distance from the target, so that

the agent will not try to stay glued to the target while trying to shoot it. This punishment is computed using Fomrula 3.8:

$$R = \begin{cases} \frac{d-d'}{50} \cdot p & \text{for } d \geq d' \\ \frac{p}{d} & \text{else} \end{cases} \tag{3.8}$$

where:

R : is the total reward

d : is the distance from the agent to the target, that is clamped in the interval $[0, 50]$

d' : is the desired distance from the agent to the target

p : is the penalty for the agent not being at desired distance d' from the target

The second punishment is applied when the agent is in front of the target, which means that the target can attack it. This punishment is added to try and teach the agent that it should flank its target so that it can attack it while being safe from being shot at. The punishment is computed using Formula 3.9

$$R = (1 - \frac{\alpha}{180}) \cdot p \tag{3.9}$$

where:

R : is the total reward

α : is the angle between the target's forward vector and the vector from the target to the agent

p : is the penalty for being in front of the target

In addition to the rewards and penalties already being used, described in Table 3.1, the new ones have the following values:

3.3.3 Training

The agent's training was done in 2 ways: a naive way, where the agent is only rewarded if it manages to shoot the target, without regards with its positioning, and a more *tactical* way where the agent is punished for not keeping distance between it and the target, and also for being in front of the target. These two approaches are documented at 3.3.3 and 3.3.3 respectively.

Name	Value	Notes
Shoot Target Reward	10	
Fire Bullet Reward	0.1	
Bullet's Trajectory is Optimal Reward	0.001	is scaled by the distance between bullet and target
Miss Target Penalty	-0.1	
Agent Not At Desired Distance Penalty	-0.005	is scaled by the difference of the desired distance and distance between agent and object
Agent In Front Of Target Penalty	-0.05	is scaled by the angle between the target's front vector and the vector from the target to the agent

Table 3.5: New Rewards and Penalites for agent that is shooting a moving target

Naive approach

As mentioned above, this approach only cares about the agent shooting the target, and not necessarily about the agent's positioning in relation to the target. This means that the penalties described in Formulae 3.8 and 3.9 are not applied. However, the reward concerning the bullet's trajectory described in Formula 3.7 is used. The observations regarding the bullet (if the bullet is fired, the bullet's trajectory vector and its speed) are used, while the observation regarding the angle between the target's front vector and vector from target to the agent is unused.

It is also worth noting that during the first trainings, the only observation regarding the bullet that was used is the one that tells if the bullet is fired.

Results for training the agent with a single network layer can be seen in Figure 3.13 and Table 3.6. As with the previous agents, the best performing configuration during training is the one with 128 units per layer, while the other two lag behind it. The configuration with 128 units was better by 32.12% than the one with 256 units, and better by 10% than the one with 512 units.

The results for training the agent with 3 network layers are in Figure 3.14 and Table 3.6. Here, the configuration with 128 units clearly overtakes the other two during most of the training process. The configuration with 128 units was better by 16.75% than the one with 256 units, and better by 41.25% than the one with 512 units at the end of the training.

Training the agent with a network with 5 layers obtains the results that can be seen in Figure 3.15 and Table 3.6. The situation is similar to the previous ones, meaning that the configuration with 128 units still performs the best. It is better by 1.76% than the one with 256 units and better by 29.38% than the one with 512 units per layer.

Results for training the agent with 7 network layers can be seen in Figure 3.16 and

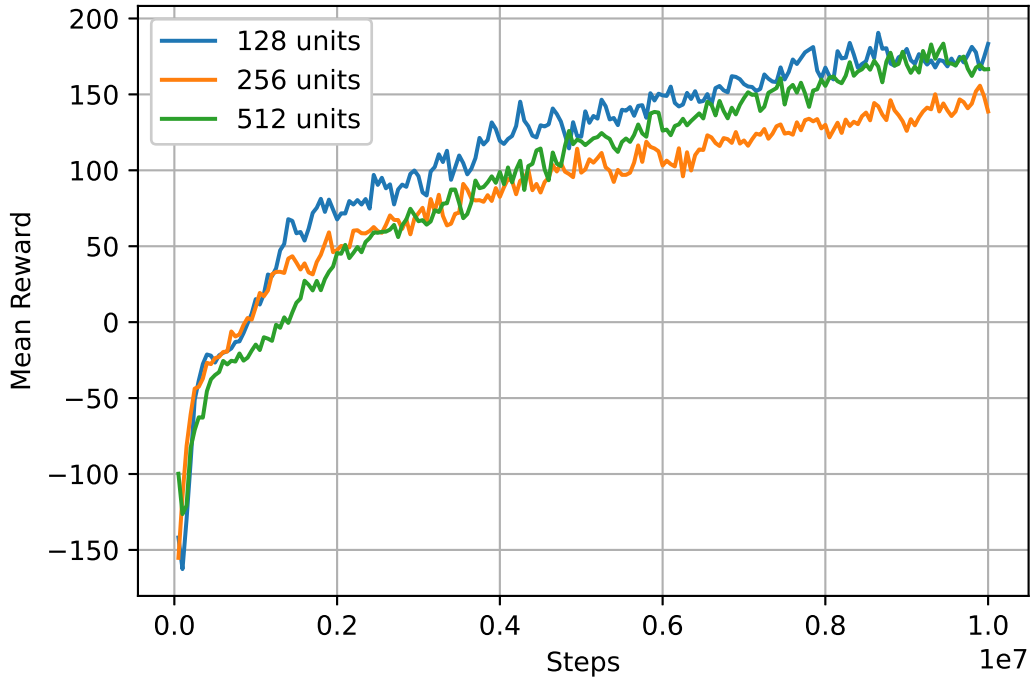

Figure 3.13: Training results for shooting a moving target with a network with 1 hidden layer

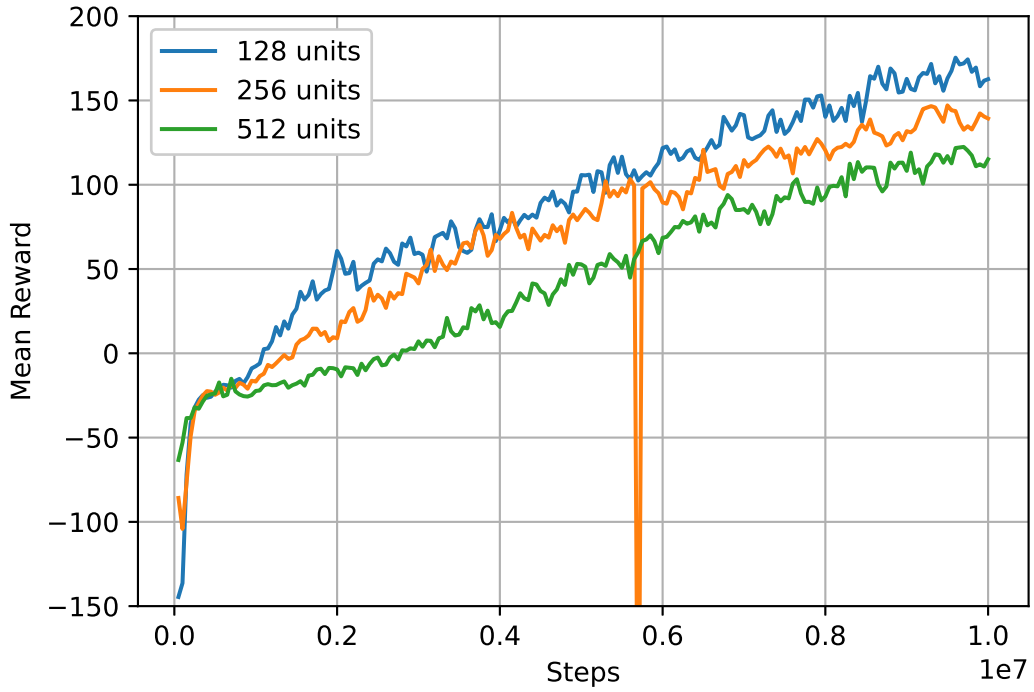

Figure 3.14: Training results for shooting a moving target with a network with 3 hidden layers

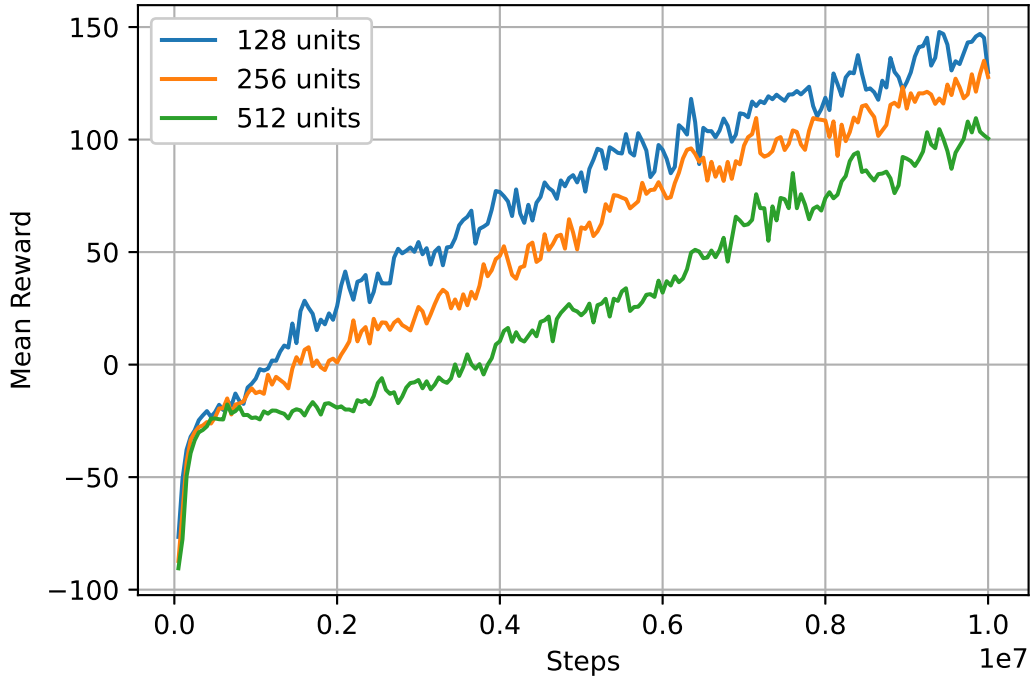

Figure 3.15: Training results for shooting a moving target with a network with 5 hidden layers

Table 3.6. Again, as in the previous cases, the configuration with the smallest number of units per layer performs better than the other ones during training. The configuration with 128 units performs better by 10.76% than the one with 256 units, and better by 29.09% than the one with 512 units per layer.

The final results for all configurations can be seen in Figure 3.17 and Table 3.6. From here we can deduce that using a smaller network with fewer units per layer gives the best results during training. Increasing the number of layers decreased the performance of the agent in every single case by up to $\sim 40\%$. Thus, it can be clearly infered that using a network architecture with 1 layer and 128 units per layer should give the best performance of the agent, and also use the least amount of computing resources.

In the second part of the trainings, the agent is trained using a configuration of 128 units per network layer, since this yielded the best results in the previous phase, and different observations regarding the bullet. The first observation combination uses all three bullet observations: if the bullet is fired, its trajectory vector and its speed; the second observation combination contains only two bullet observations: the bullet's speed and if the bullet is fired; and the final combination contains only the observation that tells if the bullet was fired. This comparasion is made to see if adding more observations would increase the agent's performance, or decrease it, similar to what happened when

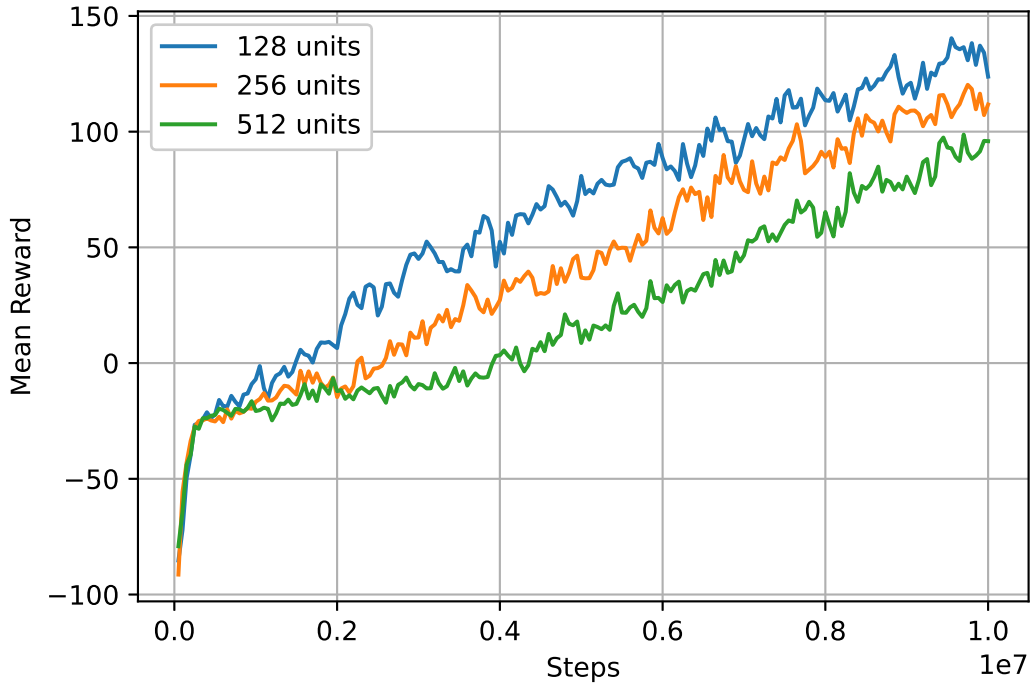

Figure 3.16: Training results for shooting a moving target with a network with 7 hidden layers

Network Configuration	**Final Mean Reward**
1 layer, 128 units	183.278
1 layer, 256 units	138.719
1 layer, 512 units	166.612
3 layers, 128 units	162.629
3 layers, 256 units	139.293
3 layers, 512 units	115.129
5 layers, 128 units	130.032
5 layers, 256 units	127.781
5 layers, 512 units	100.497
7 layers, 128 units	123.777
7 layers, 256 units	111.752
7 layers, 512 units	95.882

Table 3.6: Final training results for shooting a moving target

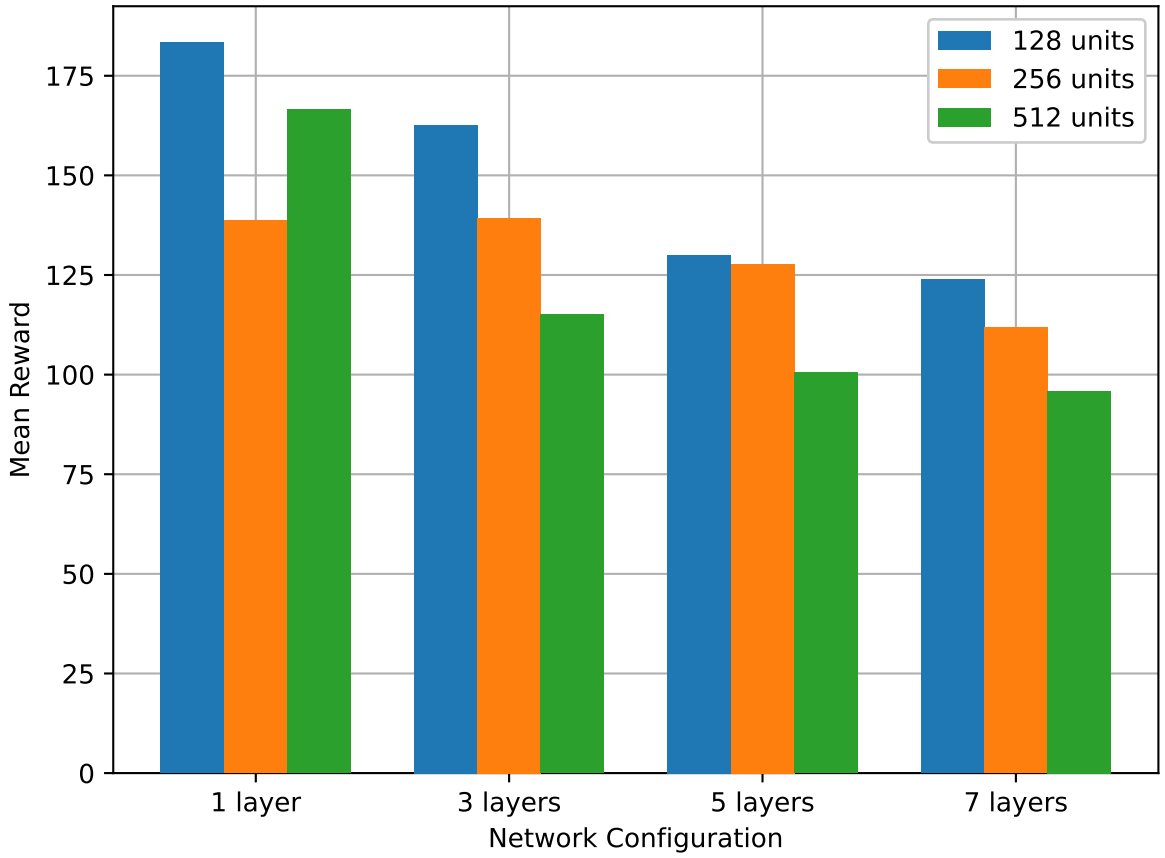

Figure 3.17: Training results for all network configurations for shooting a moving target

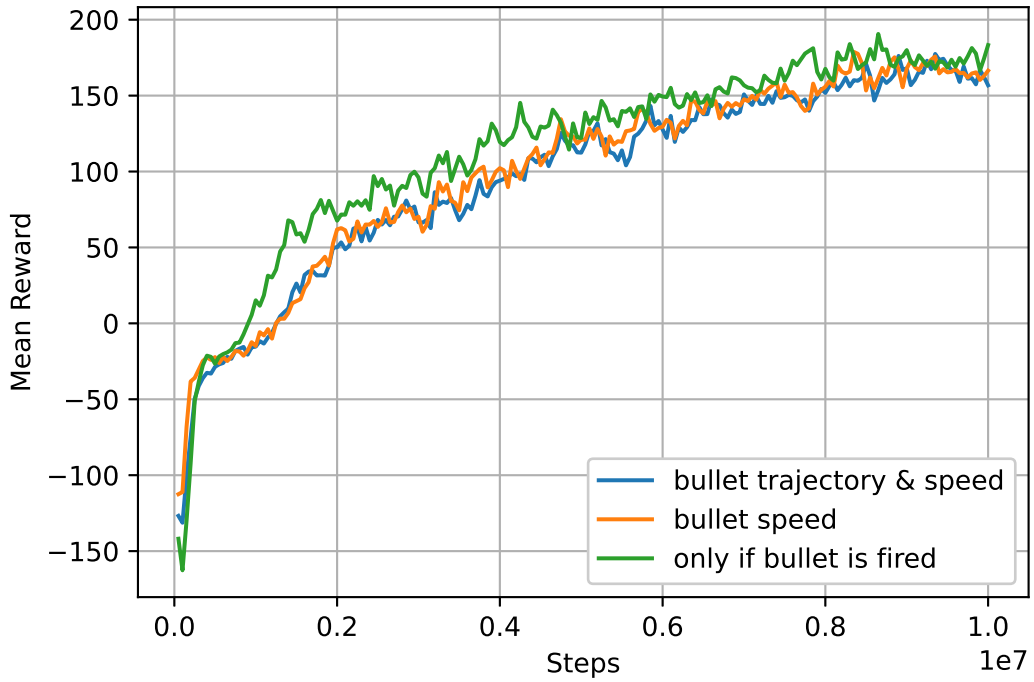

Figure 3.18: Training results for shooting a moving target with a network with 1 hidden layer, 128 units, and different observations

adding the target's movement direction to the observations in Section 3.2.3.

Results for training the agent with a single network layer can be seen in Figure 3.18 and Table 3.7. All 3 observation combinations yield very similar results, with the variant that observes only if the bullet is fired being slightly better then the other two. It is better by 16.91% than the observation combination with both bullet trajectory and speed, and better by 10.12% than the combination with the bullet's speed.

The results for training the agent with 3 network layers are in Figure 3.19 and Table 3.7. Here, it is similar to the previous situation, with all 3 combinations achieving similar training results. Here, the combination with the bullet's speed is the best one, being better by 0.41% than the combination that observes only if the bullet is fired, which means these two combinations obtain virtually identical results, and better by 7.3% than the combination with both bullet trajectory and speed.

Training the agent with a network with 5 layers obtains the results that can be seen in Figure 3.20 and Table 3.7. Again, training results are very similar for all 3 combinations. Here, the best one is the combination with both bullet trajectory and speed being better by 11.69% than the combination with the bullet's speed, and better by 18.01% the the one that only observes if the bullet was fired.

Results for training the agent with 7 network layers can be seen in Figure 3.21 and Table 3.7. All 3 combinations perform virtually identically having a maximum difference

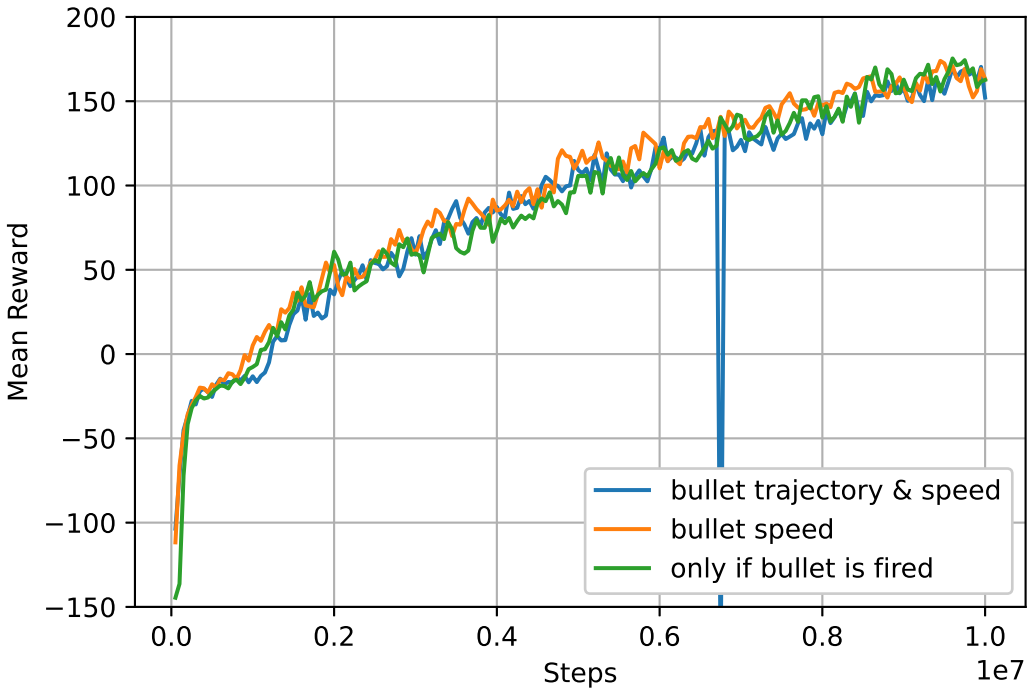

Figure 3.19: Training results for shooting a moving target with a network with 3 hidden layers, 128 units, and different observations

Figure 3.20: Training results for shooting a moving target with a network with 5 hidden layers, 128 units, and different observations

Figure 3.21: Training results for shooting a moving target with a network with 7 hidden layers, 128 units, and different observations

in agent performance of $\sim 3\%$.

The final results for all configurations can be seen in Figure 3.22 and Table 3.7. As with the first training stage, increasing the number of layers leads to a poorer performing agent. Also, there were no huge differences when introducing the new bullet observations, with the new observation combinations performing the same or slightly worse then when observing only if the bullet was fired. The only exception is when using a network with 5 layers, where adding the bullet's trajectory and speed to the observations obtains a performance that is 18.01% better than if not including them. However, the best performing configuration still remains the one with 1 layer, and where the only observation about the bullet is if it was fired.

In conclusion, when comparing the two approaches of adding or not multiple bullet observations, it seems that the best results are obtained when observing only if the bullet was fired. When the number of layers increseas, it looks like there is a tendency to obtain better results when adding more bullet observations. However, these results are still worse than when using a network architecture with 1 hidden layer, 128 units per layer, and the only bullet observation is if it was fired.

Network Configuration	Bullet Observations	Final Mean Reward
1 layer, 128 units	Bullet's trajectory and speed	156.766
1 layer, 128 units	Bullet's speed	166.427
1 layer, 128 units	Only if bullet was fired	183.278
3 layers, 128 units	Bullet's trajectory and speed	152.188
3 layers, 128 units	Bullet's speed	163.311
3 layers, 128 units	Only if bullet was fired	162.629
5 layers, 128 units	Bullet's trajectory and speed	153.46
5 layers, 128 units	Bullet's speed	137.391
5 layers, 128 units	Only if bullet was fired	130.032
7 layers, 128 units	Bullet's trajectory and speed	125.061
7 layers, 128 units	Bullet's speed	127.677
7 layers, 128 units	Only if bullet was fired	123.777

Table 3.7: Final training results for shooting a moving target with different bullet observations

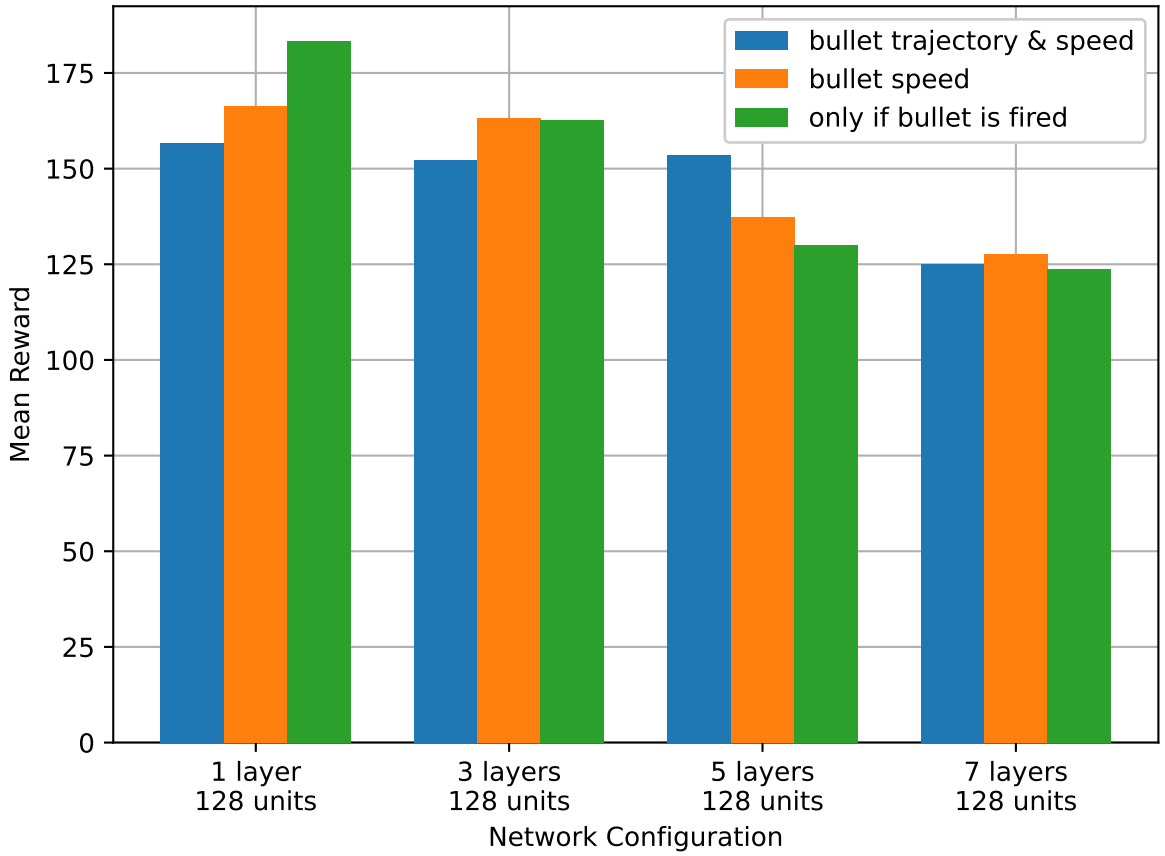

Figure 3.22: Training results for network configurations with 128 units per layer and different bullet observations, for reaching a moving target

Tactical approach

This approach tries to mimic a more realistic fighting scenario, where the target would shoot back at the agent. Thus, through the newly added punishments, described in Formulae 3.8 and 3.9, the agent should learn to keep a certain distance from the target, and also to flank it. Here, the only bullet observation that is made, is if the bullet was fired or not. This decision was made based on the training results obtained when training the agent in a more *naive* way, in the above experiment. Also, introducing more observations would require the agent to train for a larger number of steps to efficiently use them, if possible, and since the number of training steps still remains 10^7, and new penalties were added, this would lead to a poorer performing agent.

Chapter 4

Test Results

To understand how well the trained agent performs, training results are not enough since they only provide information about how well rewarded the agent is, and not about how it will perform in a real environment. For this, several tests were set up for the 3 types of problems that were tackled in this paper.

4.1 Reaching a static target

For reaching a static target, the test consists of an agent that has to touch 40 targets that are static (i. e. not moving) and that appear in a predefined order. Once the agent touches the target, it disappears and a new one appears in a new location, according to the predefined order. For each test, the same predefined order of targets is used so that the test course is the same for each run.

Initially, this test was done using only agents that were trained to reach static targets, however a more robust experiment would be to also run the test using agents that were trained to reach moving targets, and compare the results at the end.

4.1.1 Agent trained to reach static targets

The results for testing the agent that was trained to reach static targets can be seen in Table 4.1 and Figure 4.1. It can be observed that, unlike the training results, using a smaller network architecture does not yield better times, with the architecture with 5 hidden layers and 128 units per layer being the one that achieved the best time, 170.9822 seconds. For the architecure with a single hidden layer, the configuration with 128 units was faster by 22.66 seconds (11.01%) than the one with 512 units per layer. Since the configuration with 256 units per layer was unable to be trained, because it learned just to move in a circle, it was unable to finish the test course.

For the architecture with 3 hidden layers, the results were much more similar, with the configurations that had 256 and 512 units per layer, having virtually identical times, with

Network Configuration	Time to complete (s)
1 layer, 128 units	183.1415
1 layer, 256 units	DNF
1 layer, 512 units	205.8087
3 layers, 128 units	182.1884
3 layers, 256 units	180.4801
3 layers, 512 units	180.2796
5 layers, 128 units	170.9822
5 layers, 256 units	186.4627
5 layers, 512 units	205.8265
7 layers, 128 units	188.0436
7 layers, 256 units	182.1618
7 layers, 512 units	219.4408

Table 4.1: Test results for reaching a static target using agent that was trained to reach static targets

the difference between them being 0.2 seconds, and the configuration with 128 units being slower than these two by 2 seconds ($\sim 1\%$ slower). These results are somewhat surprising since there was a differnce of 29% in the training performance of the configuration with 128 units compared to the one with 512 units.

The architecture with 5 hidden layers has the configuration that is the best performing in this test, the one with 128 units per layer, which manages to achieve a time of 170.9822 seconds. The other two configurations perform worse, the one with 256 units being slower by 15.4805 seconds ($\sim 8.3\%$ slower), and the one with 512 units being slower by 34.8443 seconds ($\sim 16.92\%$ slower). This shows that a smaller number of layers does not necessarily lead to better results, however, using less units per hidden layer usually means that the agent will perform better.

The final network architecture, the one with 7 layers has the configuration with 256 units be the fastest one in the test, clocking in at 182.1618 seconds. The configuration with 128 units was slower by 5.8818 seconds (3.22%), and the one with 512 units was slower by 37.279 seconds (20.46%).

In summary, from this test we can see that increasing the number of hidden layers used in the architecture does not necessarily decrease the agent's performance, when using up to 5 layers. Also, using a smaller number of units per layer tends to lead to better agent performance. This could be due to the fact that, according to [4], increasing the width of the network can lead to overfitting.

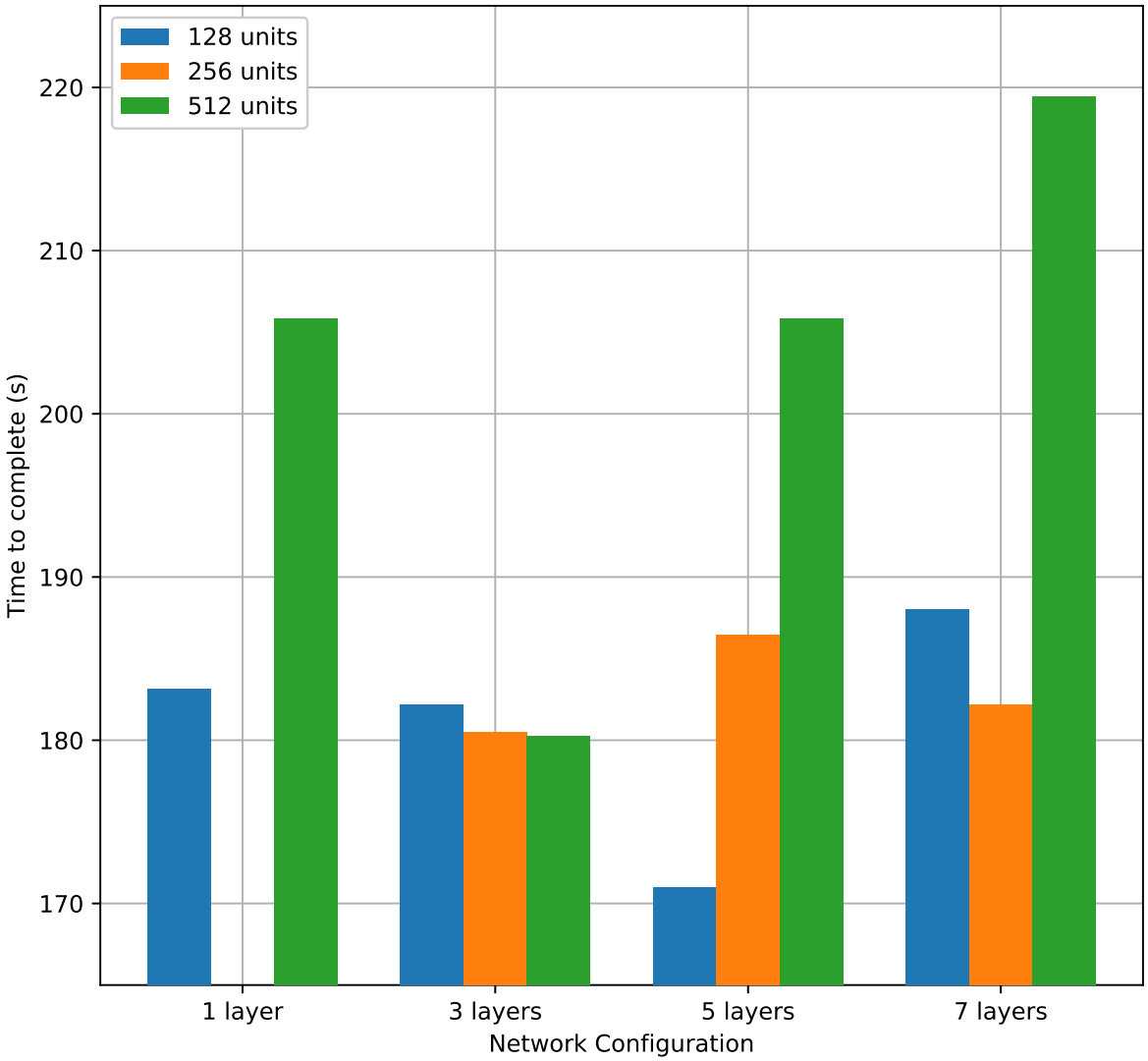

Figure 4.1: Test results for reaching a static target using agent that was trained to reach static targets

4.1.2 Agent trained to reach moving targets

The second part of the test is measuring how well an agent that was trained to reach moving targets can also reach static targets, and if it could also beat the agent that was trained to reach static targets. The obtained results are in Table 4.2 and Figure 4.2. Just like in the training Section 3.2.3, half of the tested agents will also have the target's direction as an observation (which is the vector $(0, 0, 0)$ since the target is not moving).

Looking at the results for the architecture with just a single hidden layer, it apperas that the best performing configurations are the ones with 128 units per layer, with the one without the target's direction observation being slightly faster, by 3.5389 seconds (1.8%). The configuration with 256 units per layer and with the target's direction observation takes 464.985 seconds to finish the test, more than twice the amount of time it took with 128 units per layer, due to the fact that it becomes stuck trying to move through a narrow path and can't decide to go through, because it tries to find an alternative path that does not exist. For the configurations with 512 units, the one where the target's direction is observed is faster than the other one by 7.2391 seconds (3.36%).

For the architecture with 3 hidden layers, the agents that observed the target's direction were faster than the others that did not. The fastest configuration was the one with 512 units per layer with the time of 188.5515 seconds. The second best was the one with 128 units which was slower by 3.6596 seconds (1.9%). The agents that had 256 and 512 units per layer and did not observe the target's direction, had very slow times, of 491.0297 seconds and 602.9445 seconds respectively. This was due to the fact that the agent became somehow confused and did not manage to get around the level's geometry.

The architecture with 5 hidden layers has both the slowest and fastest configuration, for this type of agent and test, at the same time. The fastest configuration was the one with 128 units and with the target's direction observation, and it finished the test course in 187.1074 seconds. The slowest configuration was the one with 512 units and without the target's direction observation, and it managed to finish the course in 576.1487 seconds. This was due to the fact that the agent was not able to correctly go around a wall and kept trying to go the wrong way for a long period of time. The configuration with 256 units and no target direction observation also managed to score a bad time of 363.128 seconds, becoming stuck in the same place as the previoulsy mentioned one.

In summary, here too, having a smaller number of units per layer seems to increase the performance of the agent, with almost all architectures having their best times be achieved by configurations with 128 units per layer. As for including the target's direction observation, it seems to positively impact the agent only when using a larger number of layers.

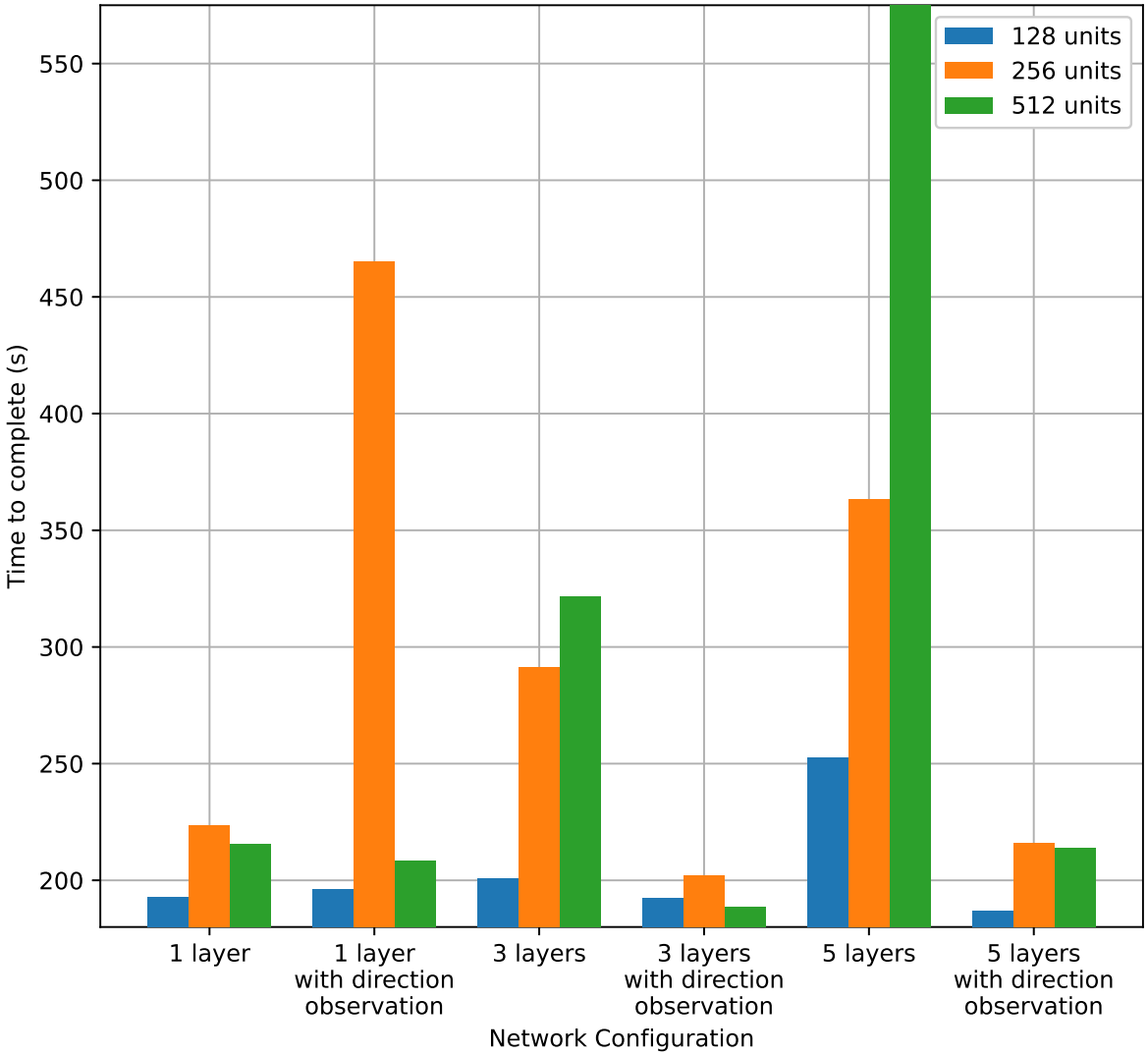

Figure 4.2: Test results for reaching a static target using agent that was trained to reach moving targets

Network Configuration	Observed target's direction	Time to complete (s)
1 layer, 128 units	No	192.7527
1 layer, 128 units	Yes	196.2916
1 layer, 256 units	No	223.6499
1 layer, 256 units	Yes	464.985
1 layer, 512 units	No	215.4364
1 layer, 512 units	Yes	208.1973
3 layers, 128 units	No	200.7101
3 layers, 128 units	Yes	192.2111
3 layers, 256 units	No	291.4421
3 layers, 256 units	Yes	202.1844
3 layers, 512 units	No	321.6053
3 layers, 512 units	Yes	188.5515
5 layers, 128 units	No	252.7738
5 layers, 128 units	Yes	187.1074
5 layers, 256 units	No	363.128
5 layers, 256 units	Yes	215.8631
5 layers, 512 units	No	576.1487
5 layers, 512 units	Yes	214.0343

Table 4.2: Test results for reaching a static target using agent that was trained to reach moving targets

4.1.3 Results comparasion

Comparing the obtained results from the two approaches, the better one was when using an agent that was trained to reach static targets. The best time for this agent was 170.9822 seconds, while the best time obtained by the agent that was trained to reach moving targets was 187.1074 seconds, which is 16.1252 seconds slower (8.61%). It is worth noting that both of these results came from architectures that had 5 hidden network layers and 128 units per layer.

The fact that the best result for reaching a static target was achieved by an agent that was trained to reach static targets is not surprising, since it was trained for this exact specific scenario.

4.2 Reaching a moving target

For reaching a moving target, the test consists of an agent that has to touch 100 moving targets that appear in a predefined order and that have predefined paths that they will take. When a target appears it moves between predefined points using a predefined route, which is different for each of the 100 times the target appears. Once the agent reaches the target, the target appears at a different location, according to the predefined order,

and the agent is reset to the middle of the level, so that it will start from the same place each times it chases a new target. This is done to make the test course be identical for each tested agent.

Just like in the previous section (4.1), the test will be run for both agents that were trained to reach moving targets and agents that were trained to reach static targets, and afterwards the results will be compared.

4.2.1 Agent trained to reach moving targets

The first ones to be tested will be the agents trained to reach moving targets. Both agents that do not observe the target's movement direction and agents that do are tested. The obtained results are in Table 4.3 and Figure 4.3.

For the architecture with a single hidden layer, the agents that did not observe the target's direction performed better than those that did, with the fastest configuration being the one with 512 units per layer, achieving a time of 452.052 seconds. It is followed by the configuration with 128 units, that was slower by 30.4473 seconds (6.73%). The configuration with 256 units and that had the target's movement direction observation performed the worst out of all agents with a time of 688.592 seconds, over 3 minutes slower than the fastest configuration due to the fact that it took suboptimal paths when chasing the target and trying to avoid the objects in the environment in a weird manner.

Looking at the results for the architecture with 3 hidden layers, the agents that had the target's direction observation performed slightly better than the agents that did not have it. The best configuration is the one with 512 units per layer, which finished the course in 467.162 seconds. The configurations 128 and 256 units, with or without the extra observation, obtained similar results that were in the range of 484–491 seconds, which means that the difference between them was of at most $\sim 1\%$. The configuration with 512 units that did not have the extra observation performed much worse, achieving a time of 602.9445 seconds, due to fact that it tried to avoid the objects placed in the level at a much greater distance than it should have.

The architecture with 5 hidden layers had very different results between the agents that used the target's movement direction observation and those that did not, whith the ones that did, performing much better, on average being faster by ~ 58 seconds. The best configuration was the one with 128 units per layer, obtaining a time of 462.4213 seconds, with the other configurations lagging behind by over 38 seconds (at least 8.2% slower).

In summary, it seems that the extra observation improves the agent's performance only when multiple hidden layers are used, otherwise it hinders its performance. The surprise here is that the fastest configuration had only one network layer, compared to the results from Section 4.1, where the best results were obtained by agents that had a network with

Network Configuration	Observed target's direction	Time to complete (s)
1 layer, 128 units	No	482.4993
1 layer, 128 units	Yes	499.2063
1 layer, 256 units	No	495.7267
1 layer, 256 units	Yes	688.592
1 layer, 512 units	No	452.052
1 layer, 512 units	Yes	485.6294
3 layers, 128 units	No	484.7428
3 layers, 128 units	Yes	488.6792
3 layers, 256 units	No	491.0297
3 layers, 256 units	Yes	485.8719
3 layers, 512 units	No	602.9445
3 layers, 512 units	Yes	467.162
5 layers, 128 units	No	527.4669
5 layers, 128 units	Yes	462.4213
5 layers, 256 units	No	549.7428
5 layers, 256 units	Yes	500.7562
5 layers, 512 units	No	568.0677
5 layers, 512 units	Yes	506.722

Table 4.3: Test results for reaching a moving target using agent that was trained to reach moving targets

5 hidden layers. Here, the difference between the best achieved time, form the agent with 1 hidden layer, and the second best time, from the agent with 5 hidden layers, is of 10.3693 seconds, which means that the second best agent was slower by 2.29%.

4.2.2 Agent trained to reach static targets

Now, an agent that was trained to reach a static target will be tested to see how fast it will reach a moving target, and afterwards compare the results with the ones obtained by the agents that were trained to reach moving targets. The obtained results are in Table 4.3 and Figure 4.3.

Looking at the results for the architecture with a single hidden layer, it can be seen that the best result, out of all the agents tested, is obtained by the one with the configuration with 128 units per layer, and which finishes the test course in 427.4408 seconds. The configuration with 256 units did not finish the course since it was unable to be trained. The configuration with 512 units is slower by 11.3486 seconds (2.65% slower).

For the architecture with 3 hidden layers, the configurations with 128 units and 512 units obtain results that are very similar, the only difference being of 1.8478 seconds (0.41%), with the one with 512 units being the faster one. The configuration with 256 units is slightly slower, by 8.7071 seconds (1.94%).

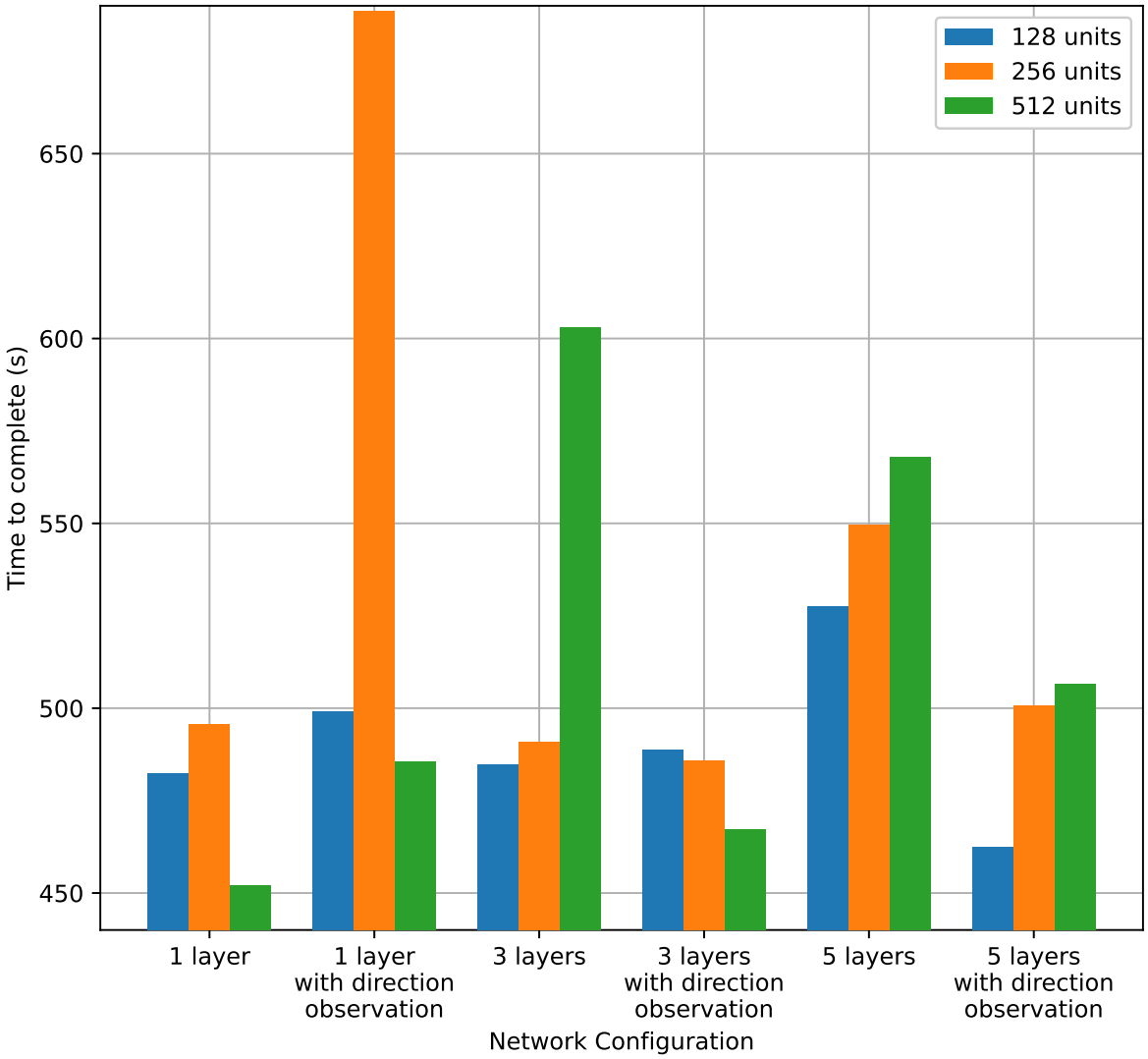

Figure 4.3: Test results for reaching a moving target using agent that was trained to reach moving targets

Network Configuration	Time to complete (s)
1 layer, 128 units	427.4408
1 layer, 256 units	DNF
1 layer, 512 units	438.7894
3 layers, 128 units	450.3436
3 layers, 256 units	457.2029
3 layers, 512 units	448.4958
5 layers, 128 units	429.3176
5 layers, 256 units	466.765
5 layers, 512 units	462.7256
7 layers, 128 units	465.1996
7 layers, 256 units	479.0871
7 layers, 512 units	546.0353

Table 4.4: Test results for reaching a moving target using agent that was trained to reach static targets

The architecture with 5 hidden layers has the second best performing agent, this being the one that has 128 units per layer and that finished the course in 429.3176 seconds. The other two agents managed to be slower, with the agent that had 256 units per layer being slower by 37.4474 seconds (8.72%), and the agent that had 512 units being slower by 33.408 seconds (7.78%).

The architecture with 7 layers, has the slowest results, with the best one from this architecture being the configuration with 128 units which managed to finish the test in 465.1996 seconds. The slowest configuration was the one that had 512 units per layer, finishing the course in 546.0353 seconds, 17.37% slower than the one with 128 units. Again, the agent tries to make very large turns to avoid other objects, which make him lose a lot of time.

In summary, the best agent was the one that had an architecture with 1 hidden layer and 128 units per layer, finishing the test in 427.4408 seconds, and being closely followed by the agent with the architecture with 5 hidden layers and 128 units, which was slower by 1.8768 seconds. This shows, that unlike the test in Section 4.1, having a smaller network leads to better results.

4.2.3 Results comparasion

Comparing the obtained results, it can be seen that the agent who was trained to reach static targets, performs better than the agent trained to reach moving targets. The fastest agent trained to reach static targets finished the test in 427.4408 seconds, while the fastest agent trained to reach moving targets finished the test in 452.052 seconds, being slower by 24.6112 seconds (5.75%). This means that training an agent for a particular

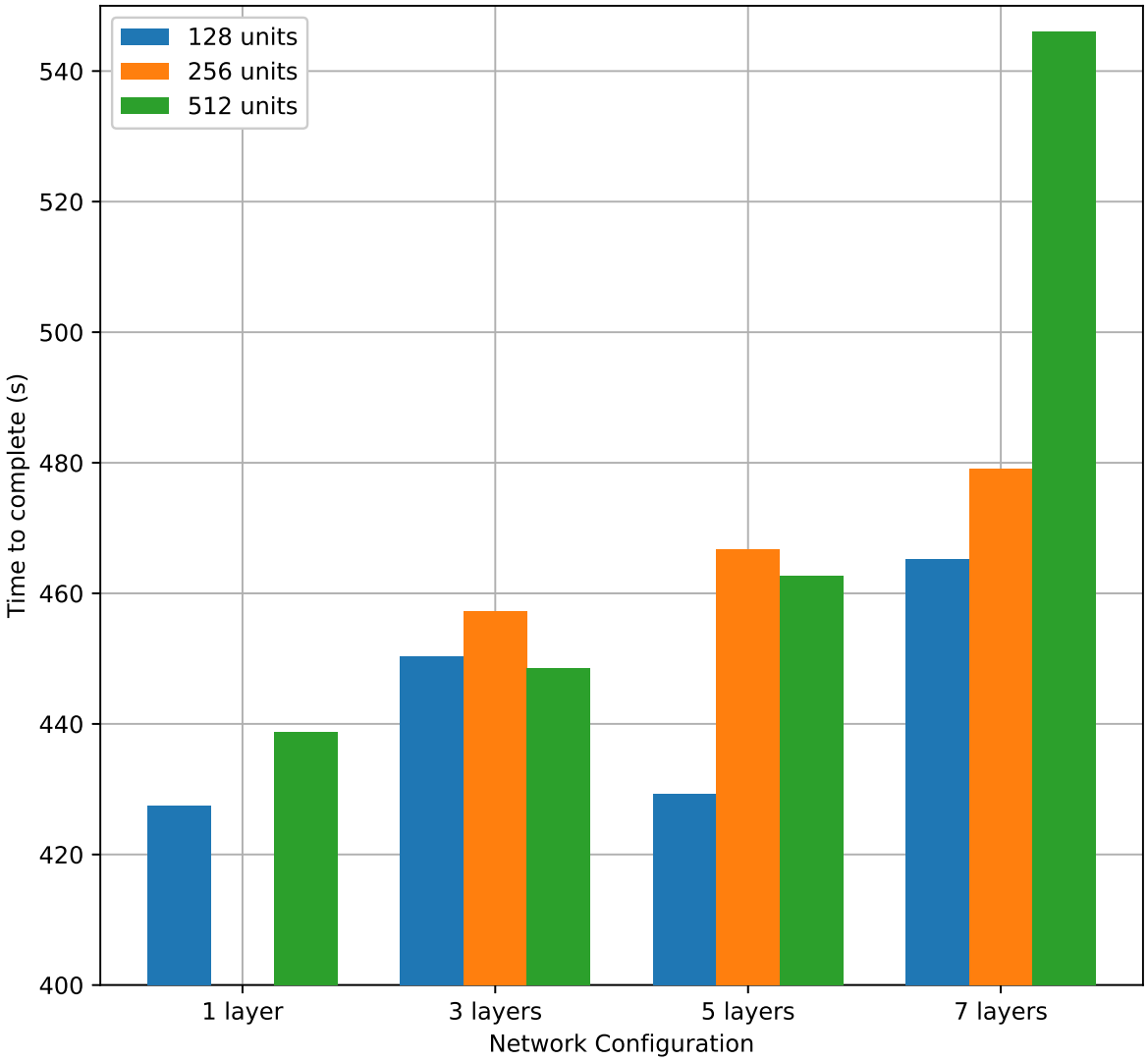

Figure 4.4: Test results for reaching a moving target using agent that was trained to reach static targets

Network Configuration	Time to complete (s)
1 layer, 128 units	499.8927
1 layer, 256 units	564.696
1 layer, 512 units	536.6353
3 layers, 128 units	590.7621
3 layers, 256 units	637.085
3 layers, 512 units	736.3587
5 layers, 128 units	606.5751
5 layers, 256 units	675.0099
5 layers, 512 units	817.632
7 layers, 128 units	837.6456
7 layers, 256 units	872.1839
7 layers, 512 units	914.4853

Table 4.5: Test results for shooting a moving target using agent that was trained using *naive* method

task could make it perform a more advanced version of that task better than an agent trained specifically for that advanced task. It is also worth noting that both agents had a network with a single hidden layer.

4.3 Shooting a moving target

For shooting a moving target, the test is built just like it is described in Section 4.2, the only difference being that the agent is not supposed to reach the target, but to shoot it. The two types of trained agents will be tested, however, comparasions between them regarding how fast can they finish the test are not as important as before, since one of the approaches is supposed to be more aggressive (the *naive* one), and the other is supposed to be more defensive (the tactical one), with the result being that the more aggressive apporach should finish the test faster.

4.3.1 Agent trained in a *naive* manner

4.3.2 Agent trained in a tactical manner

4.4 Fighting against an AI controlled enemy

TODO: sa fac sa pot sa bat un tank, dupa sa vad daca paote sa bata inamicu, si daca da, sa vad cat de repede poa sa bata inamicu de x ori (eventual sa fac un mean time per round, sau sa zic de cate ori a fost lovit, etc)

Network Configuration	Bullet Observations	Time to complete (s)
1 layer, 128 units	Bullet's trajectory and speed	528.1937
1 layer, 128 units	Bullet's speed	547.2708
1 layer, 128 units	Only if bullet was fired	499.8927
3 layers, 128 units	Bullet's trajectory and speed	529.0487
3 layers, 128 units	Bullet's speed	557.6132
3 layers, 128 units	Only if bullet was fired	590.7621
5 layers, 128 units	Bullet's trajectory and speed	628.6615
5 layers, 128 units	Bullet's speed	688.095
5 layers, 128 units	Only if bullet was fired	606.5751
7 layers, 128 units	Bullet's trajectory and speed	687.3325
7 layers, 128 units	Bullet's speed	684.9317
7 layers, 128 units	Only if bullet was fired	837.6456

Table 4.6: Test results for shooting a moving target using agent that was trained using *naive* method and different bullet observation combinations

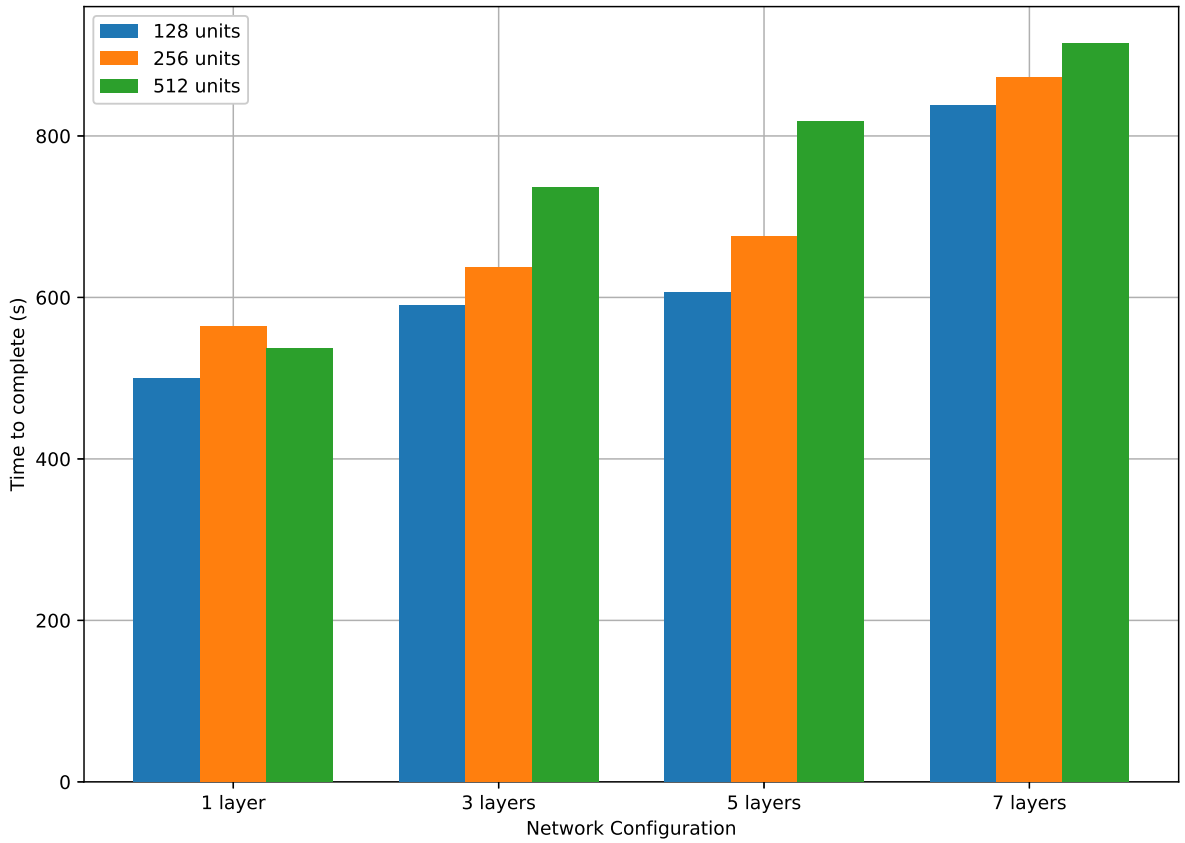

Figure 4.5: Test results for shooting a moving target using agent that was trained using *naive* method

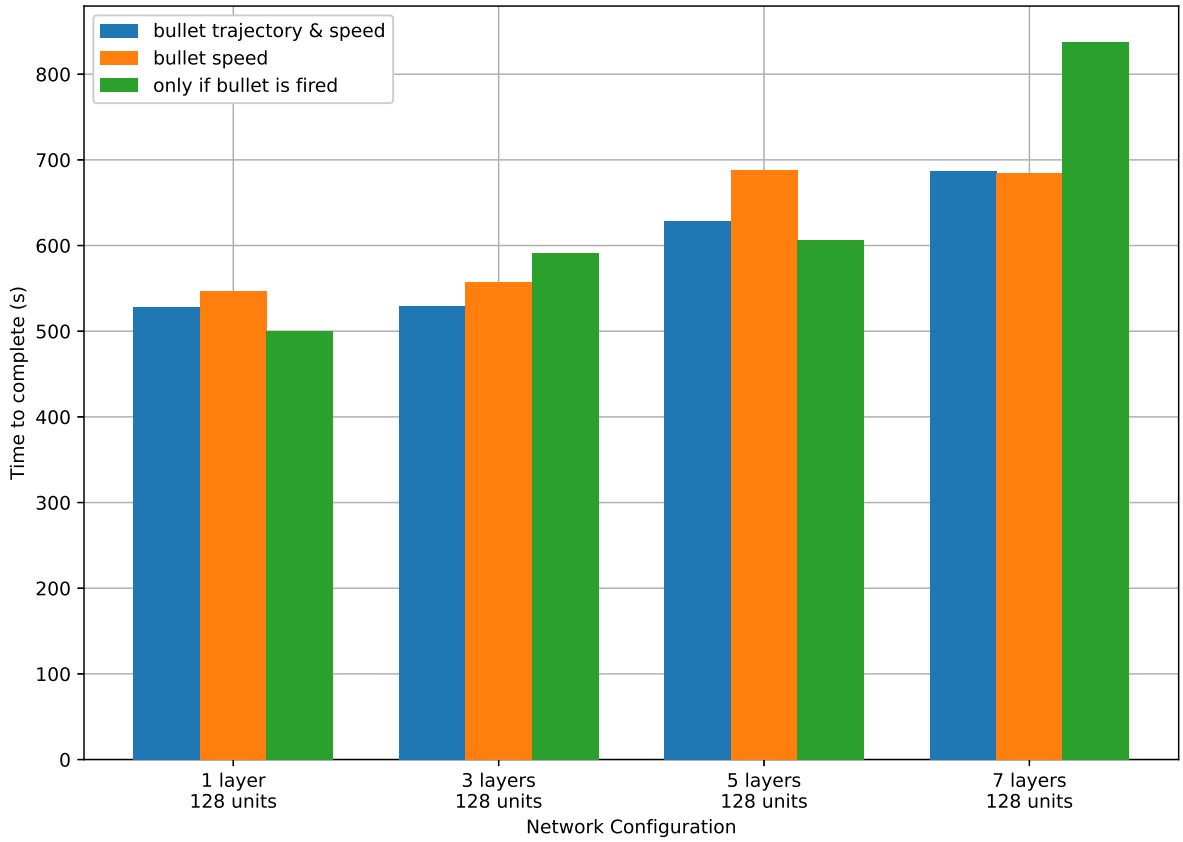

Figure 4.6: Test results for shooting a moving target using agent that was trained using *naive* method and different bullet observation combinations

Chapter 5

Conclusions

Bibliography

[1] Eloi Alonso, Maxim Peter, David Goumard, and Joshua Romoff. *Deep Reinforcement Learning for Navigation in AAA Video Games*. 2020. arXiv: 2011.04764 [cs.LG].

[2] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. "Automatic computer game balancing: a reinforcement learning approach." In: July 2005, pp. 1111–1112. DOI: 10.1145/1082473.1082648.

[3] Olivier Buffet, Olivier Pietquin, and Paul Weng. *Reinforcement Learning*. 2020. arXiv: 2005.14419 [cs.LG].

[4] Ronen Eldan and Ohad Shamir. *The Power of Depth for Feedforward Neural Networks*. 2016. arXiv: 1512.03965 [cs.LG].

[5] Thomas Krendl Gilbert, Nathan Lambert, Sarah Dean, Tom Zick, and Aaron Snoswell. *Reward Reports for Reinforcement Learning*. 2023. arXiv: 2204.10817 [cs.LG].

[6] Hung Guei and I-Chen Wu. *On Reinforcement Learning for the Game of 2048*. 2022. arXiv: 2212.11087 [cs.LG].

[7] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. *Unity: A General Platform for Intelligent Agents*. 2020. arXiv: 1809.02627 [cs.LG].

[8] Ruo-Ze Liu, Zhen-Jia Pang, Zhou-Yu Meng, Wenhai Wang, Yang Yu, and Tong Lu. *On Efficient Reinforcement Learning for Full-length Game of StarCraft II*. 2022. arXiv: 2209.11553 [cs.LG].

[9] Vasileios Moschopoulos, Pantelis Kyriakidis, Aristotelis Lazaridis, and Ioannis Vlahavas. *Lucy-SKG: Learning to Play Rocket League Efficiently Using Deep Reinforcement Learning*. 2023. arXiv: 2305.15801 [cs.LG].

[10] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. *Gotta Learn Fast: A New Benchmark for Generalization in RL*. 2018. arXiv: 1804.03720 [cs.LG].

[11] Inseok Oh, Seungeun Rho, Sangbin Moon, Seongho Son, Hyoil Lee, and Jinyun Chung. *Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning*. 2020. arXiv: 1904.03821 [cs.AI].

[12] Tim Pearce and Jun Zhu. *Counter-Strike Deathmatch with Large-Scale Behavioural Cloning*. 2021. arXiv: 2104.04258 [cs.AI].

[13] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. *A Game Theoretic Framework for Model Based Reinforcement Learning*. 2021. arXiv: 2004.07804 [cs.LG].

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[15] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. *A Survey of Deep Reinforcement Learning in Video Games*. 2019. arXiv: 1912.10944 [cs.MA].

[16] Geraud Nangue Tasse, Tamlin Love, Mark Nemecek, Steven James, and Benjamin Rosman. *ROSARL: Reward-Only Safe Reinforcement Learning*. 2023. arXiv: 2306.00035 [cs.LG].

[17] Yaodong Yang and Jun Wang. *An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective*. 2021. arXiv: 2011.00583 [cs.MA].