



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

1	Introduction	4
1.1	Lorem ipsum	4
1.2	Dolor sit amet	4
1.2.1	Quis tellus vitae	4
2	Preliminarii	5
3	Training TODO: change name, kinda cringe	6
3.1	Reaching a static target	6
3.1.1	The Problem	6
3.1.2	Implementing the solution	6
3.1.3	Training	11
3.2	Reaching a moving target	14
3.2.1	The Problem	14
3.2.2	Implementing the solution	16
3.2.3	Training	16
3.3	Shooting a moving target	22
3.3.1	The Problem	22
3.3.2	Implementing the solution	22
3.3.3	Training	22
4	Test Results	29
4.1	Reaching a static target	29
4.2	Reaching a moving target	29
4.3	Shooting a moving target	29
	Bibliography	38

Chapter 1

Introduction

1.1 Lorem ipsum

1.2 Dolor sit amet

1.2.1 Quis tellus vitae

Chapter 2

Preliminarii

Chapter 3

Training TODO: change name, kinda cringe

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 position 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 were 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).

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

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

$$R = \frac{r}{d_i} \quad (3.2)$$

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 combat 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 through 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).

$$R = \frac{p}{d_i}, \text{ if } d_i < d \quad (3.3)$$

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).

$$R = \frac{\alpha}{180} \cdot p \quad (3.4)$$

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 movements, 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).

$$R = \frac{\alpha}{180} \cdot p \quad (3.5)$$

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

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

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 position 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%.

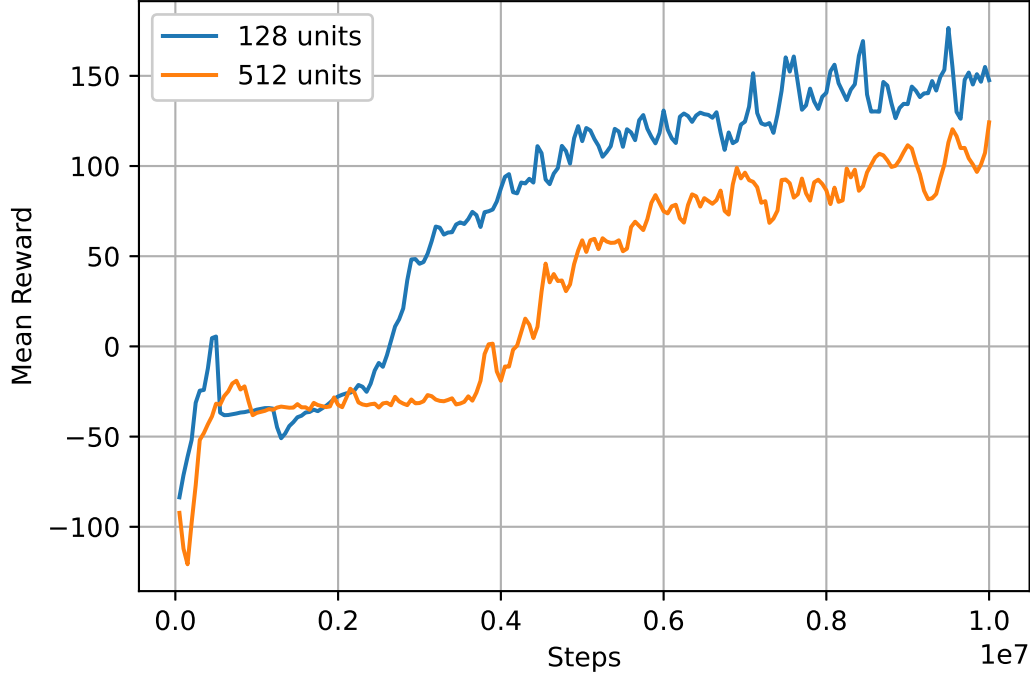


Figure 3.1: Training results for reaching a static target with a network with 1 hidden layer

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 similarly 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 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.

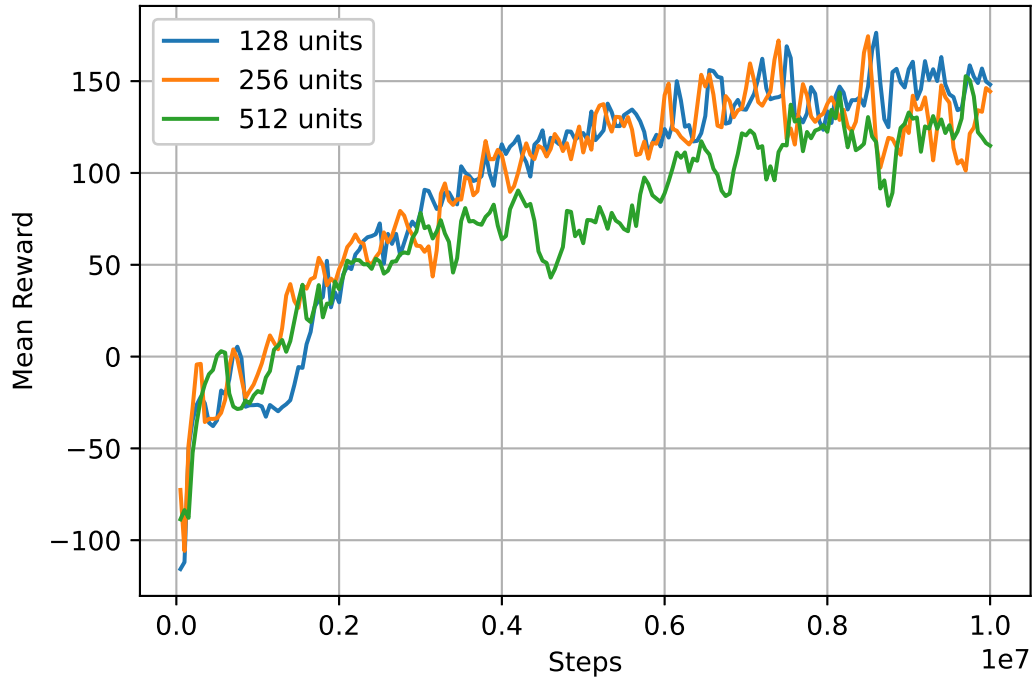


Figure 3.2: Training results for reaching a static target with a network with 3 hidden layers

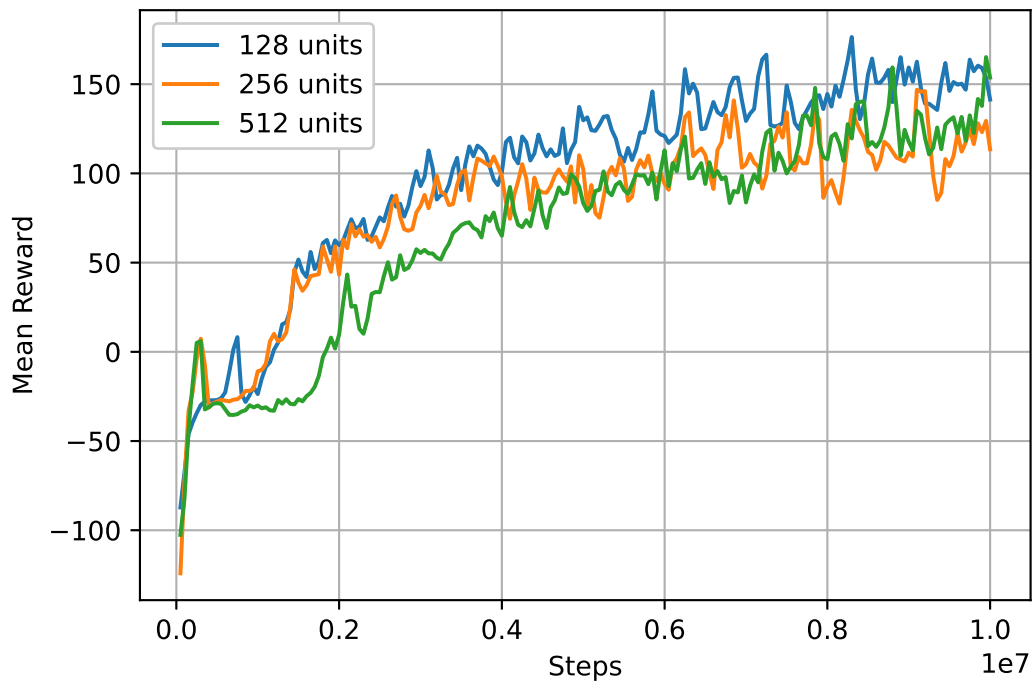


Figure 3.3: Training results for reaching a static target with a network with 5 hidden layers

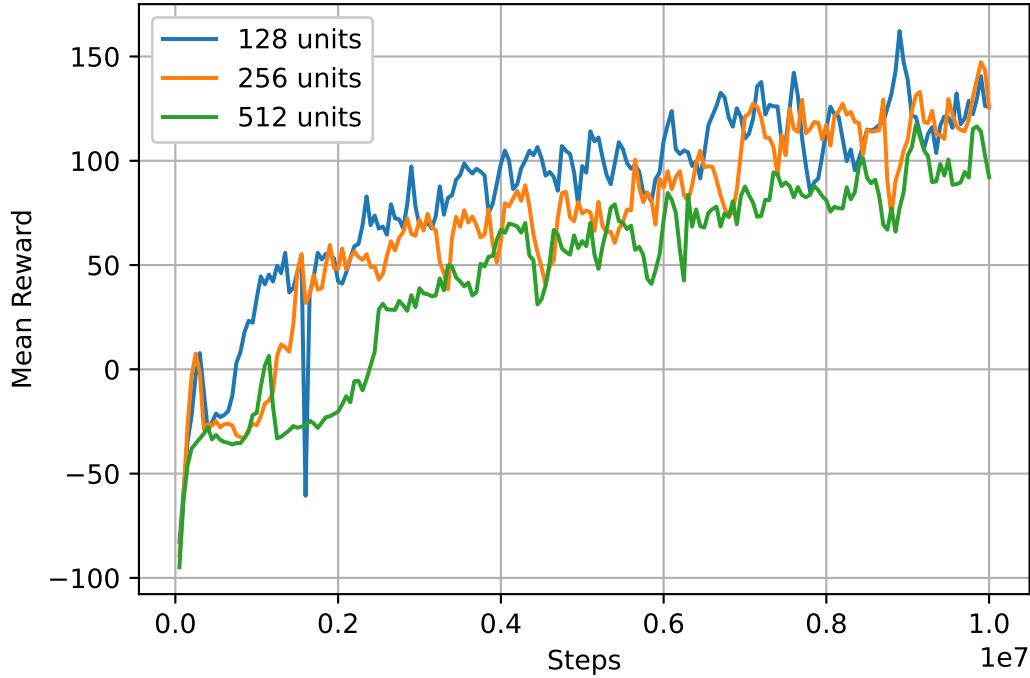


Figure 3.4: Training results for reaching a static target with a network with 7 hidden layers

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

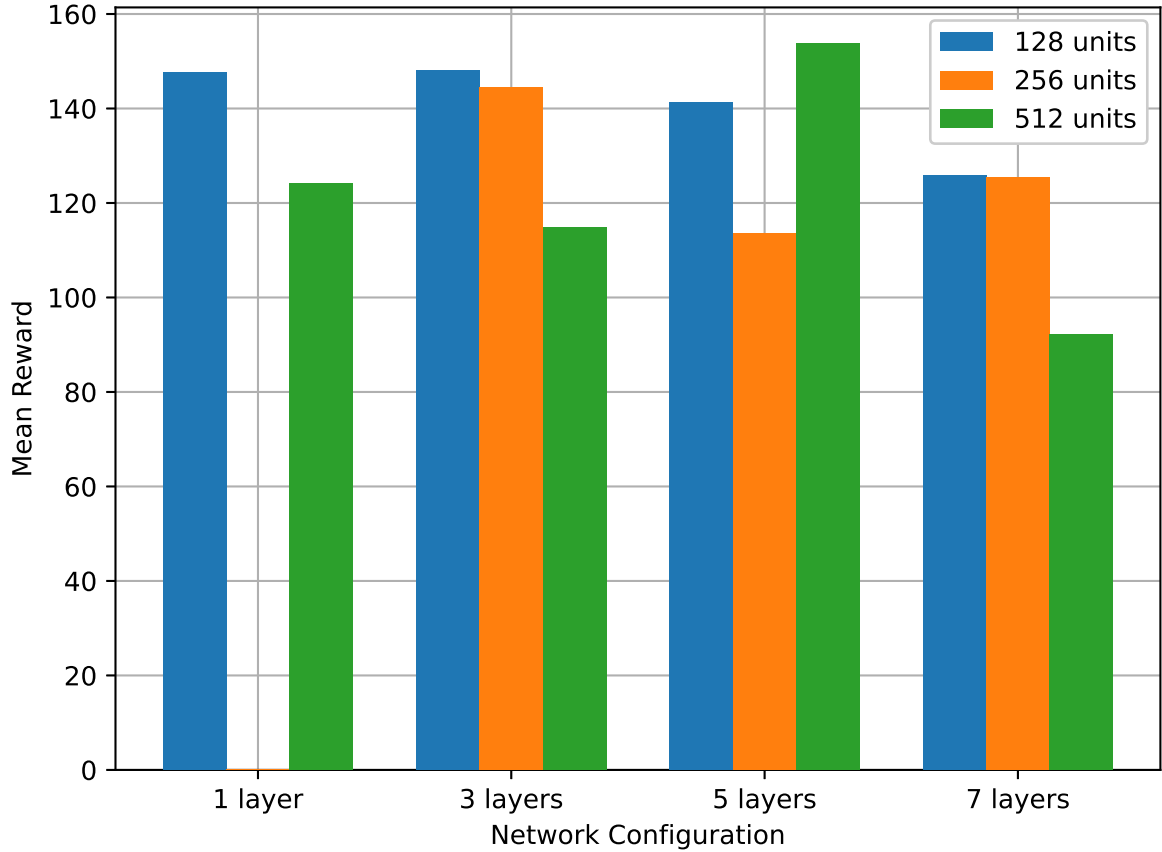


Figure 3.5: Training results for all network configurations for reaching a static target

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 observation 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 observation 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 the configuration with 256 units, which in turn obtains better results than the one with 512 units. At the

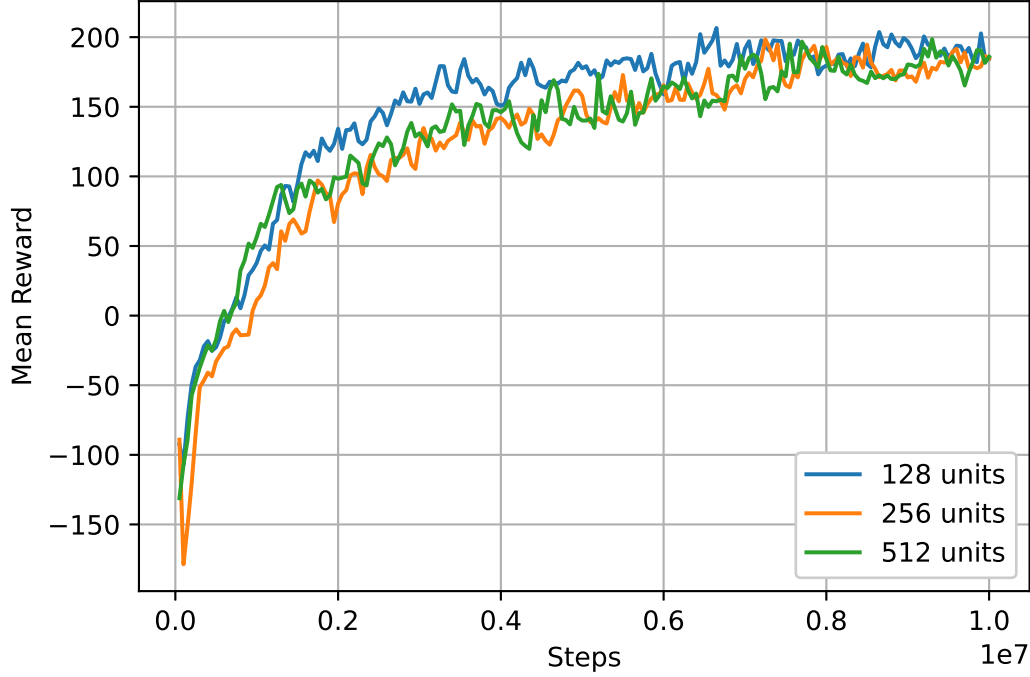


Figure 3.6: Training results for reaching a moving target with a network with 1 hidden layer

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 observation 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 observation can be seen in Figure 3.9 and Table 3.4.

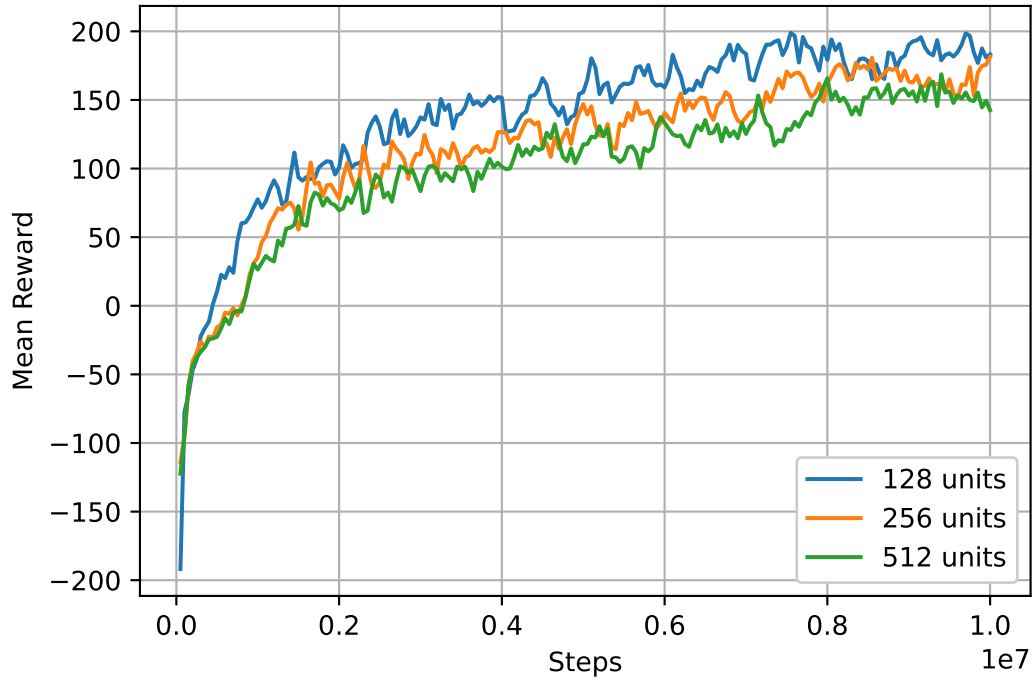


Figure 3.7: Training results for reaching a moving target with a network with 3 hidden layers

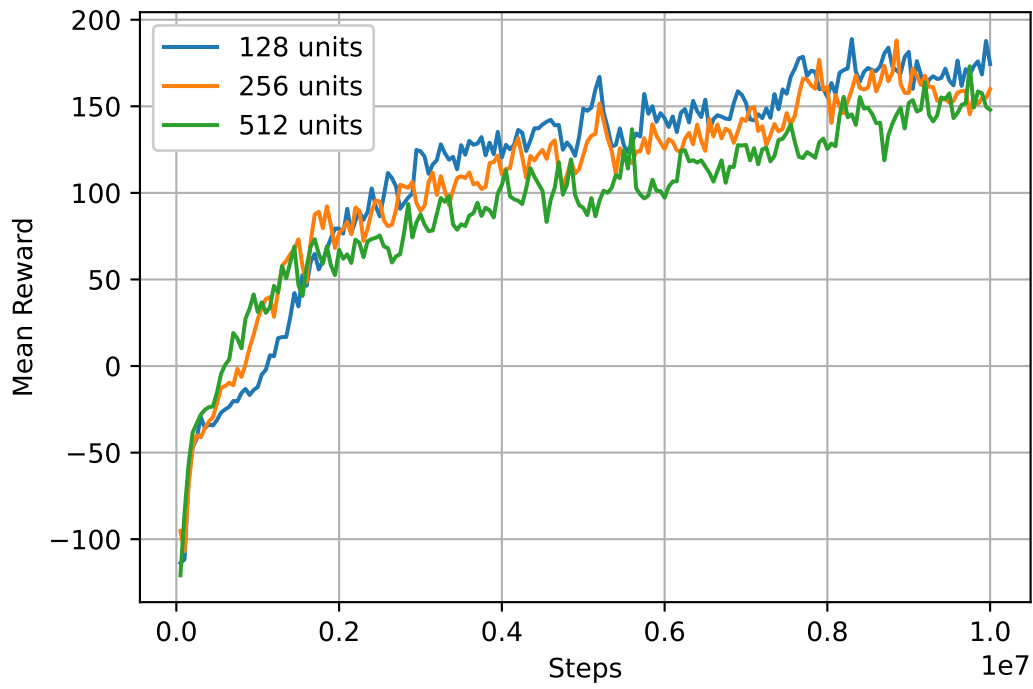


Figure 3.8: Training results for reaching a moving target with a network with 5 hidden layers

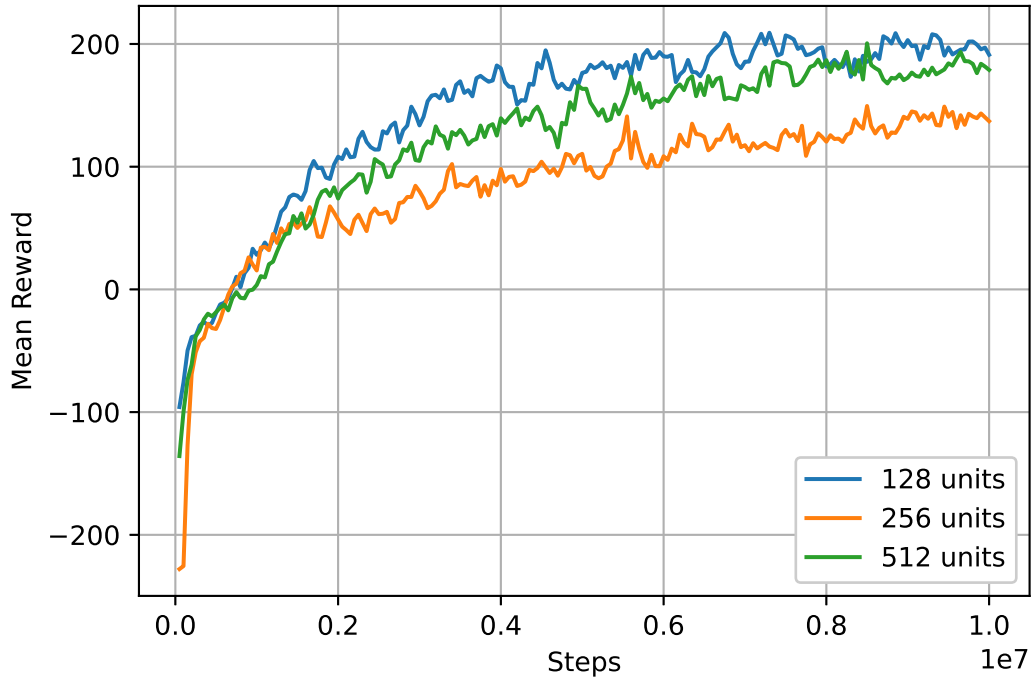


Figure 3.9: Training results for reaching a moving target with a network with 1 hidden layer and with observation of target's direction

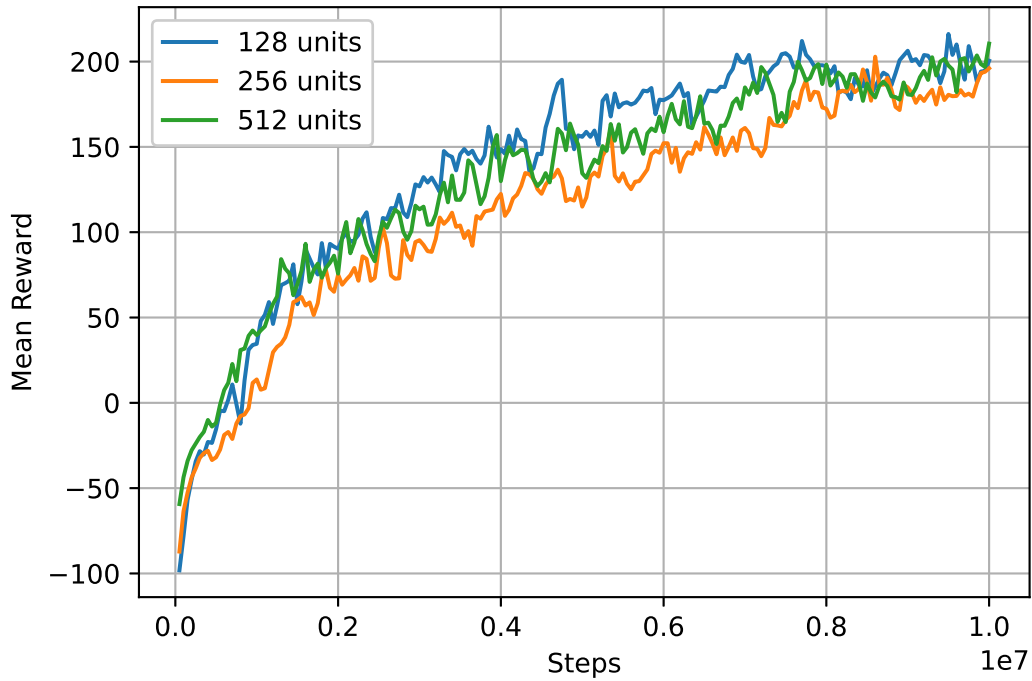


Figure 3.10: Training results for reaching a moving target with a network with 3 hidden layers and with observation of target's direction

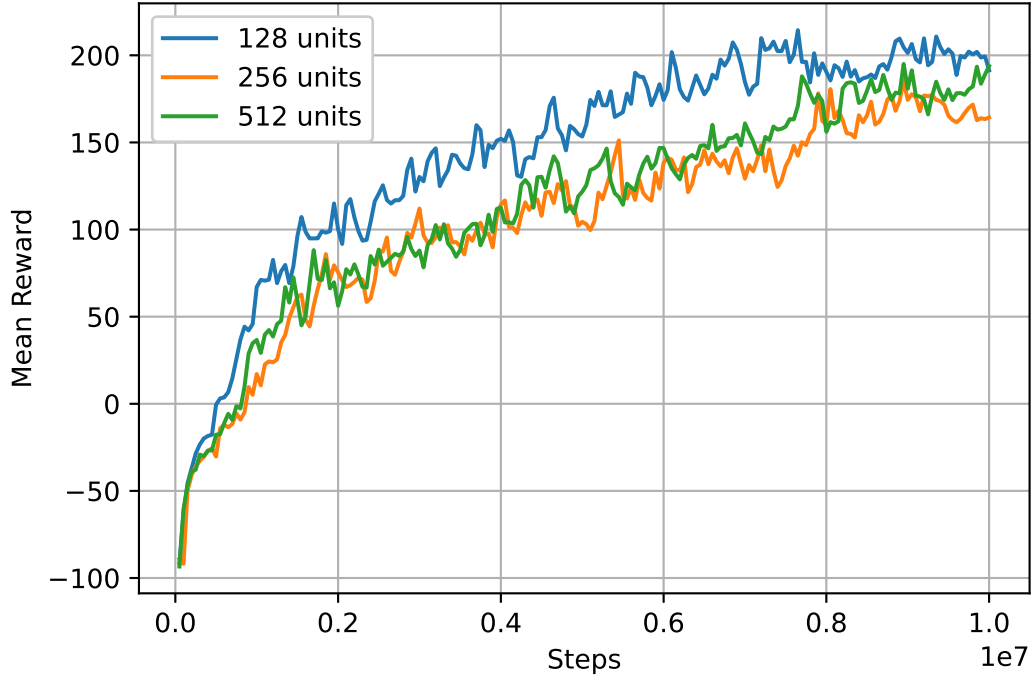


Figure 3.11: Training results for reaching a moving target with a network with 5 hidden layers and with observation of target’s direction

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

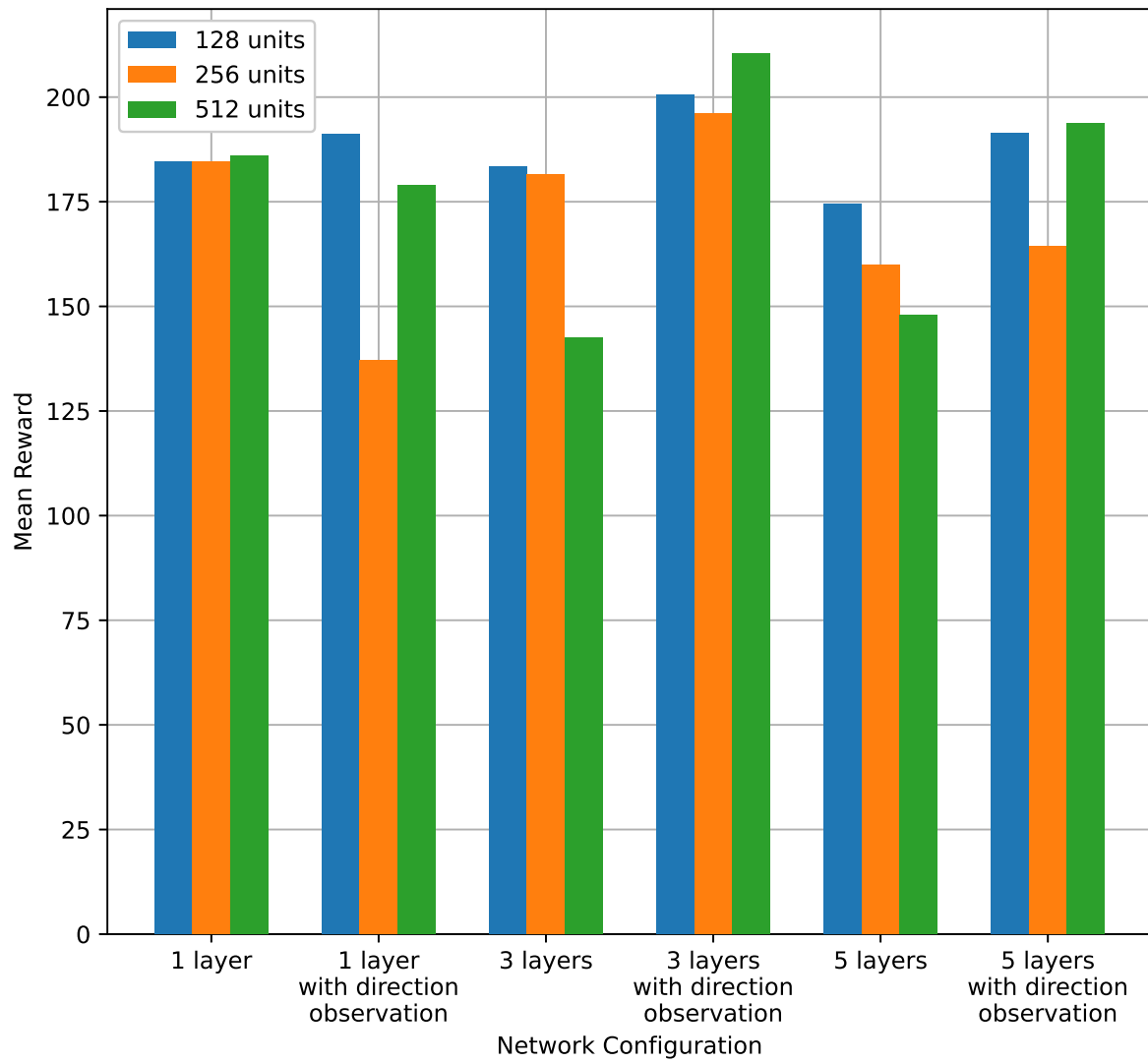


Figure 3.12: Training results for all network configurations for reaching a moving target

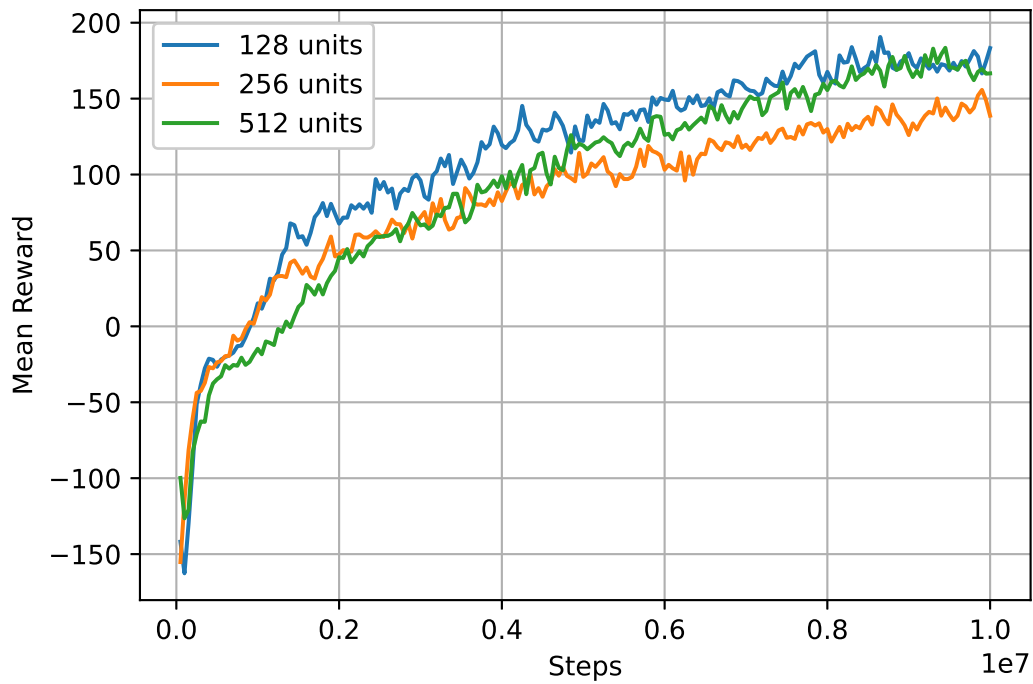


Figure 3.13: Training results for shooting a moving target with a network with 1 hidden layer

3.3 Shooting a moving target

3.3.1 The Problem

3.3.2 Implementing the solution

3.3.3 Training

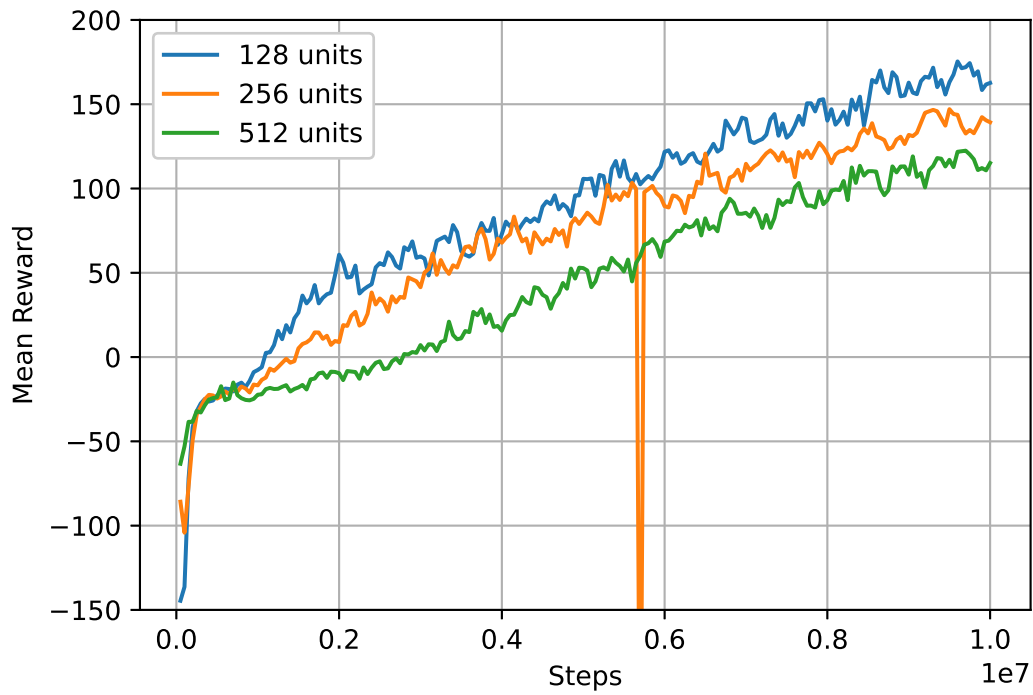


Figure 3.14: Training results for shooting a moving target with a network with 3 hidden layers

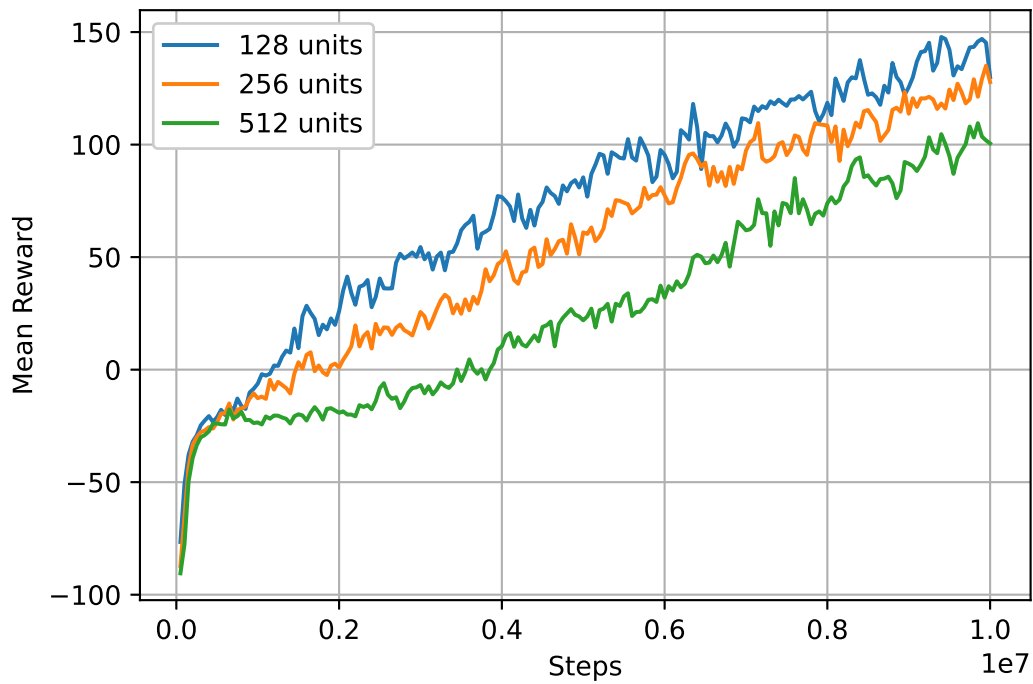


Figure 3.15: Training results for shooting a moving target with a network with 5 hidden layers

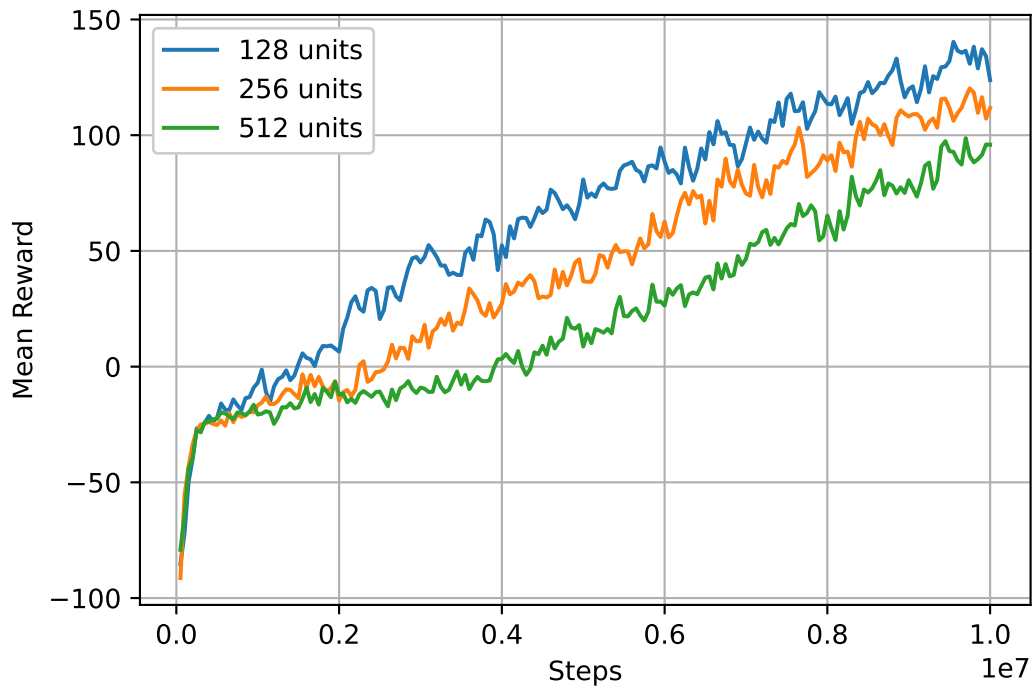


Figure 3.16: Training results for shooting a moving target with a network with 7 hidden layers

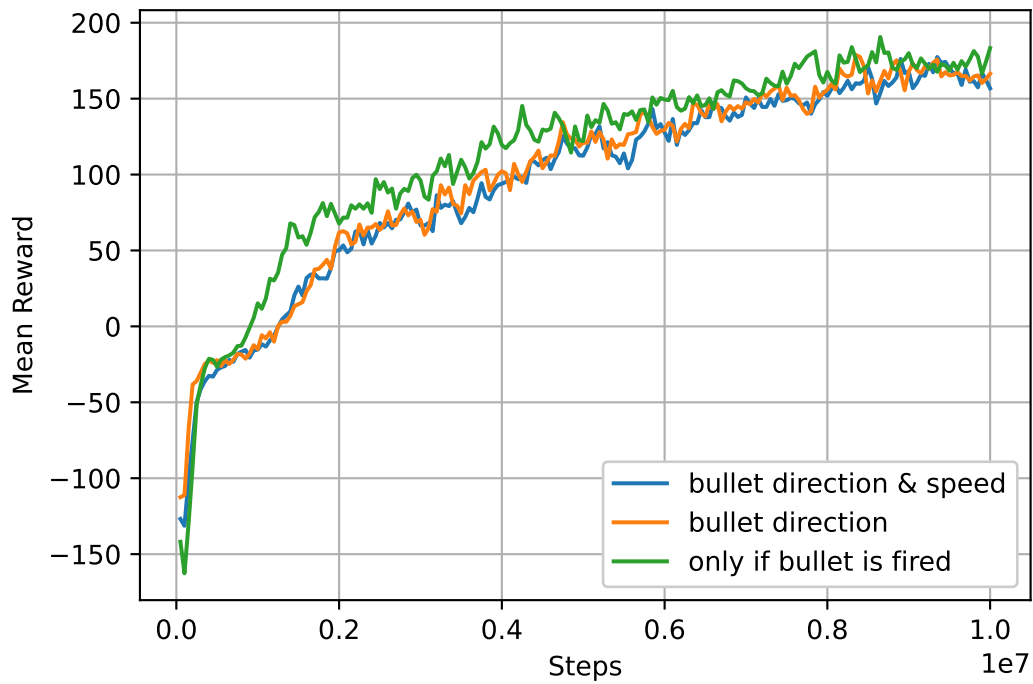


Figure 3.17: Training results for shooting a moving target with a network with 1 hidden layer, 128 units, and different observations

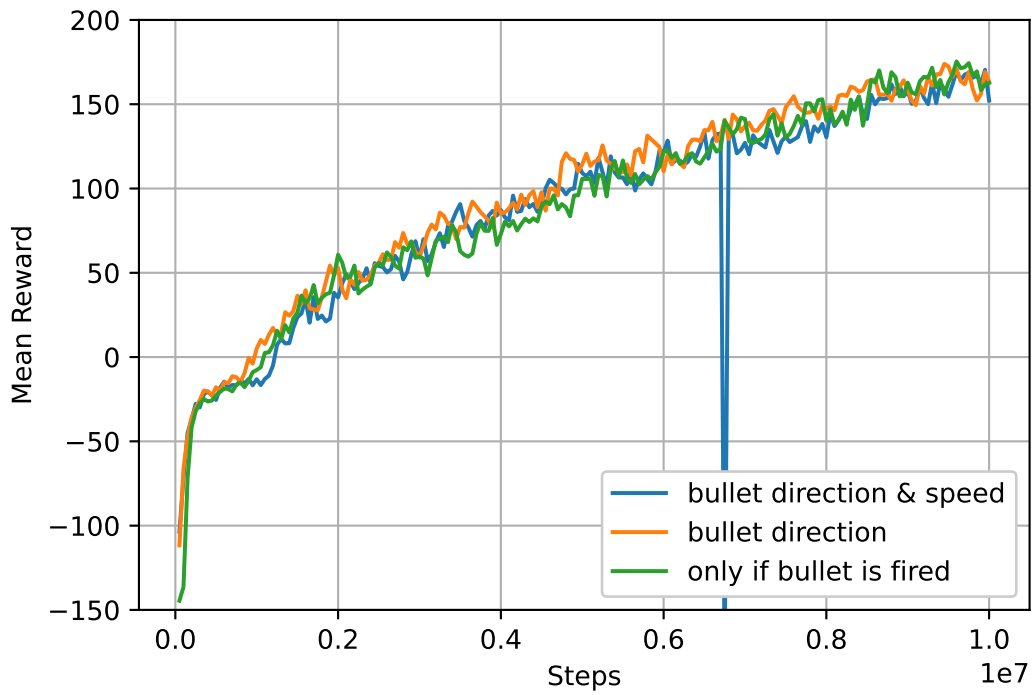


Figure 3.18: Training results for shooting a moving target with a network with 3 hidden layers, 128 units, and different observations



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

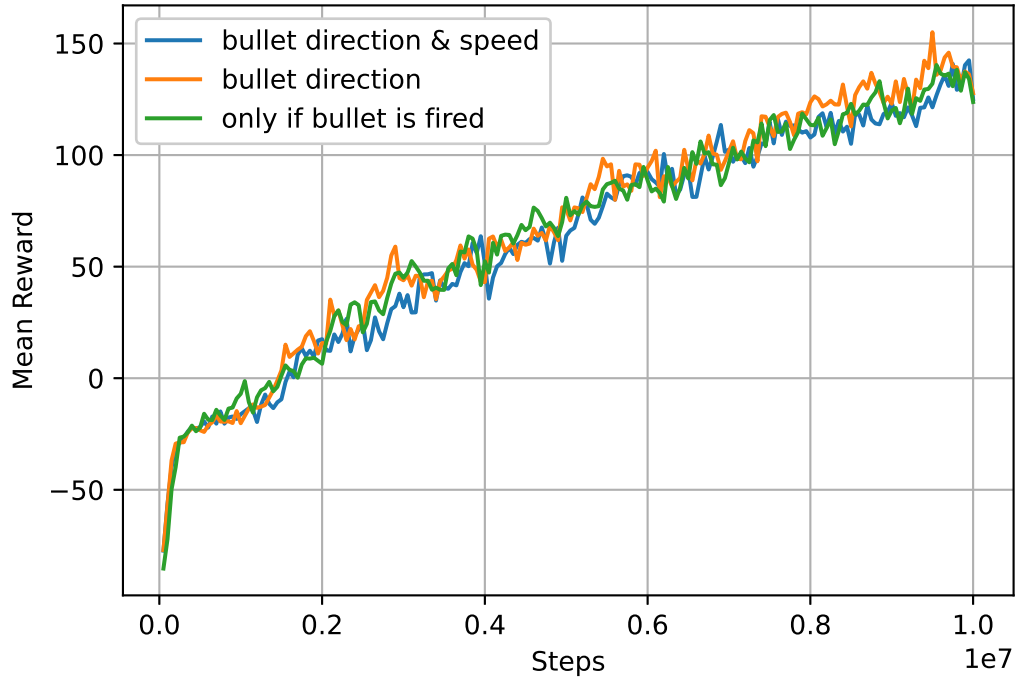


Figure 3.20: Training results for shooting a moving target with a network with 3 hidden layers, 128 units, and different observations

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 3.5: Final training results for shooting a moving target

Network Configuration	Bullet Observations	Final Mean Reward
1 layer, 128 units	Bullet's direction 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 direction 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 direction 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 direction 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 3.6: Final training results for shooting a moving target with different bullet observations

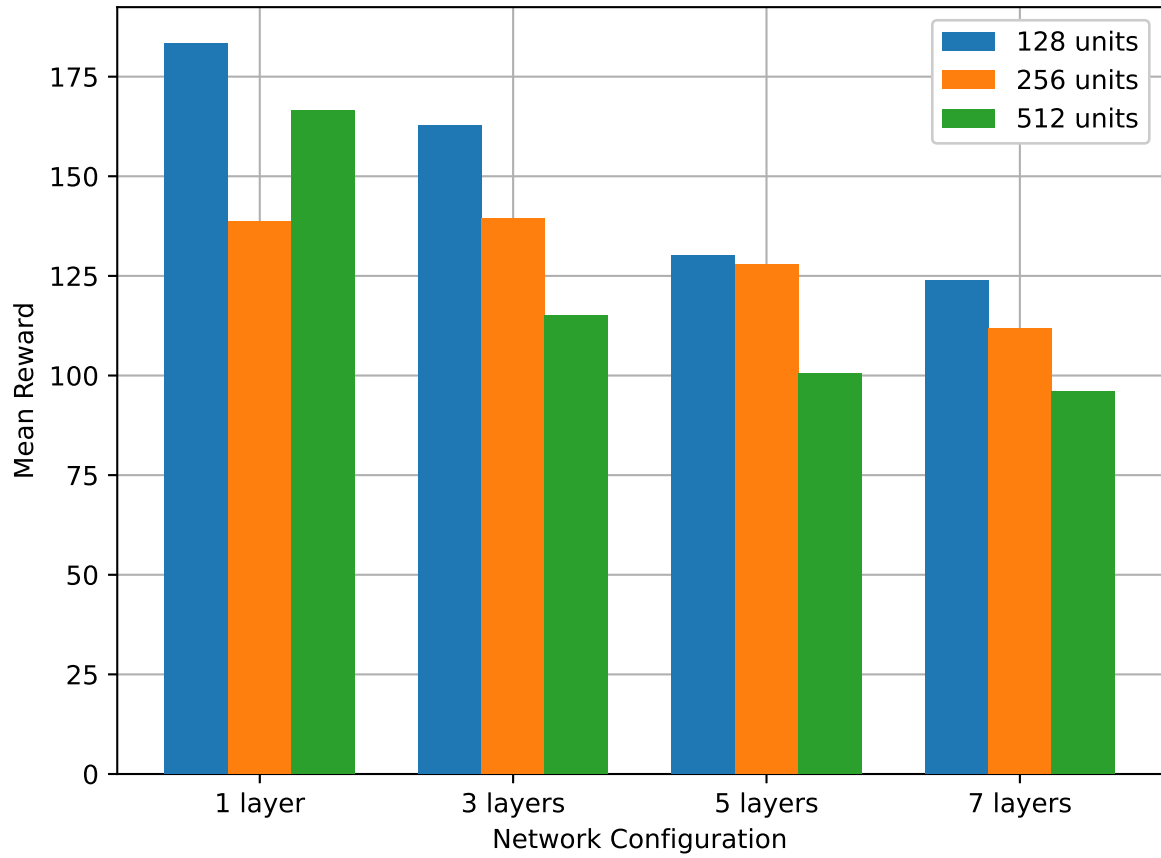


Figure 3.21: Training results for all network configurations for shooting a moving target

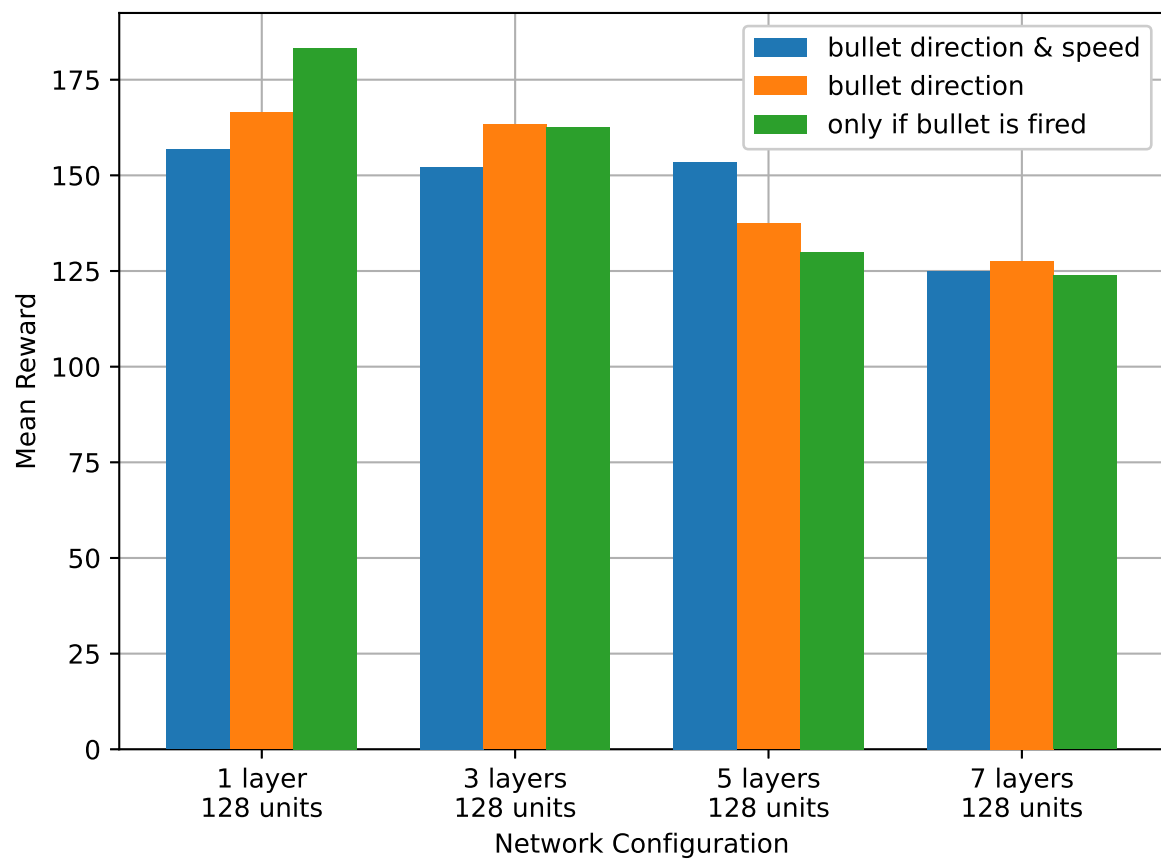


Figure 3.22: Training results for network configurations with 128 units per layer and different bullet observations, for reaching a moving target

Chapter 4

Test Results

4.1 Reaching a static target

TODO: la static target se atinge obiectivu de 40 de ori

4.2 Reaching a moving target

TODO: la moving target se atinge obiectivu de 100 de ori

4.3 Shooting a moving target

TODO: la shooting target se impusca obiectivu de 100 de ori

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 static course, static brain

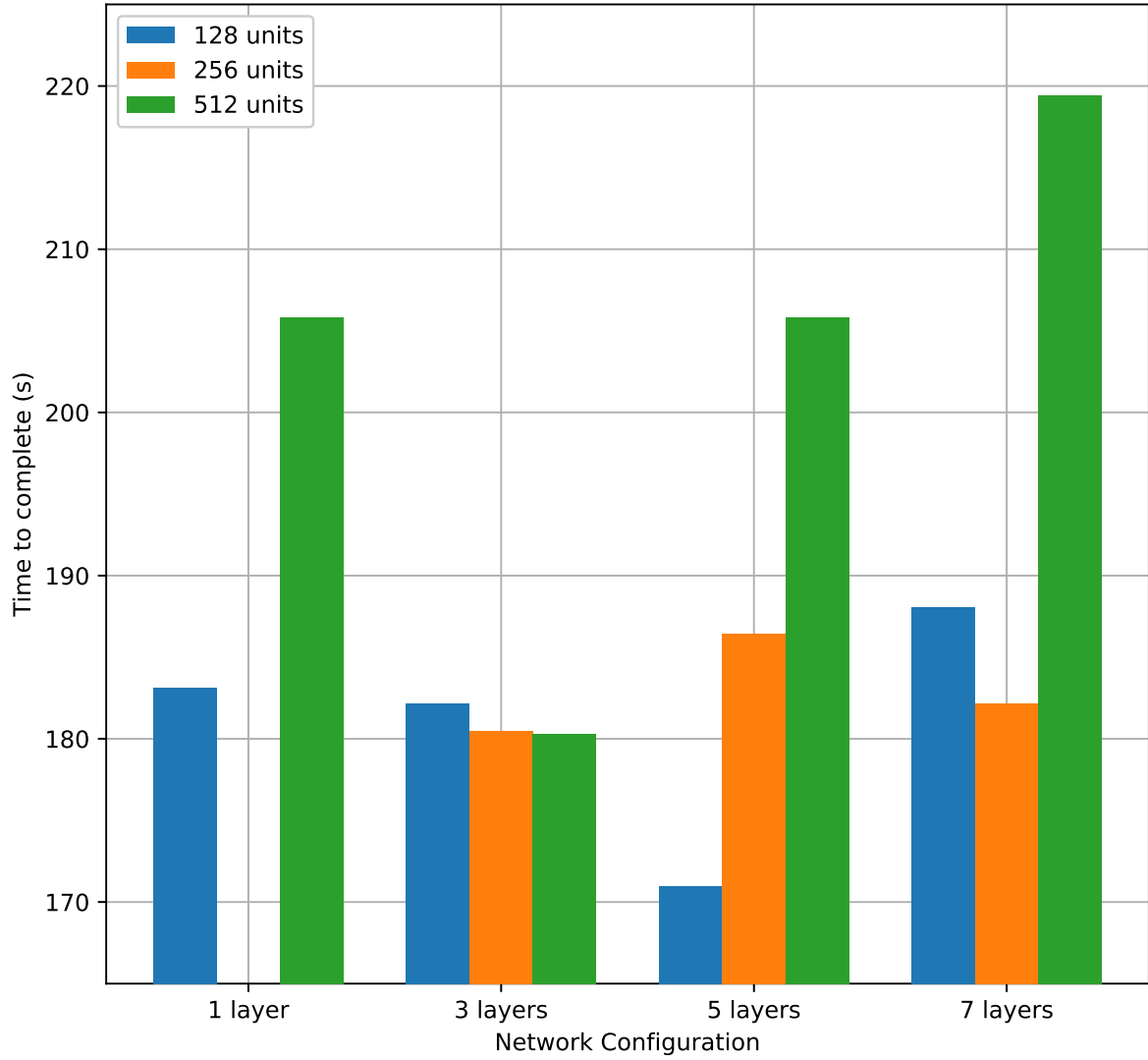


Figure 4.1: Test results for static course & static brain

Network Configuration	Observed direction target's	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 static course, moving brain

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 4.3: Test results moving course, static brain

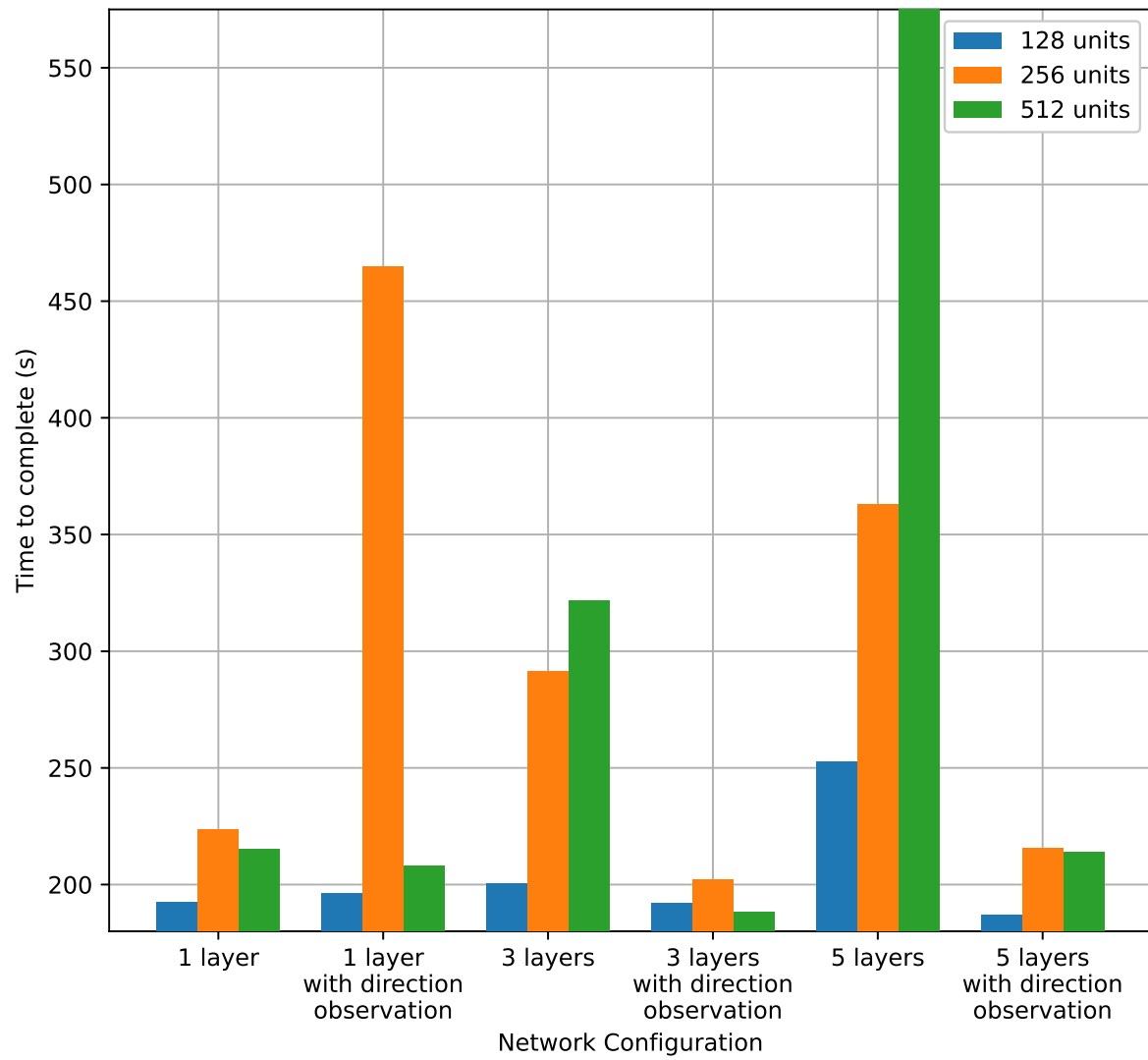


Figure 4.2: Test results for static course & moving brain

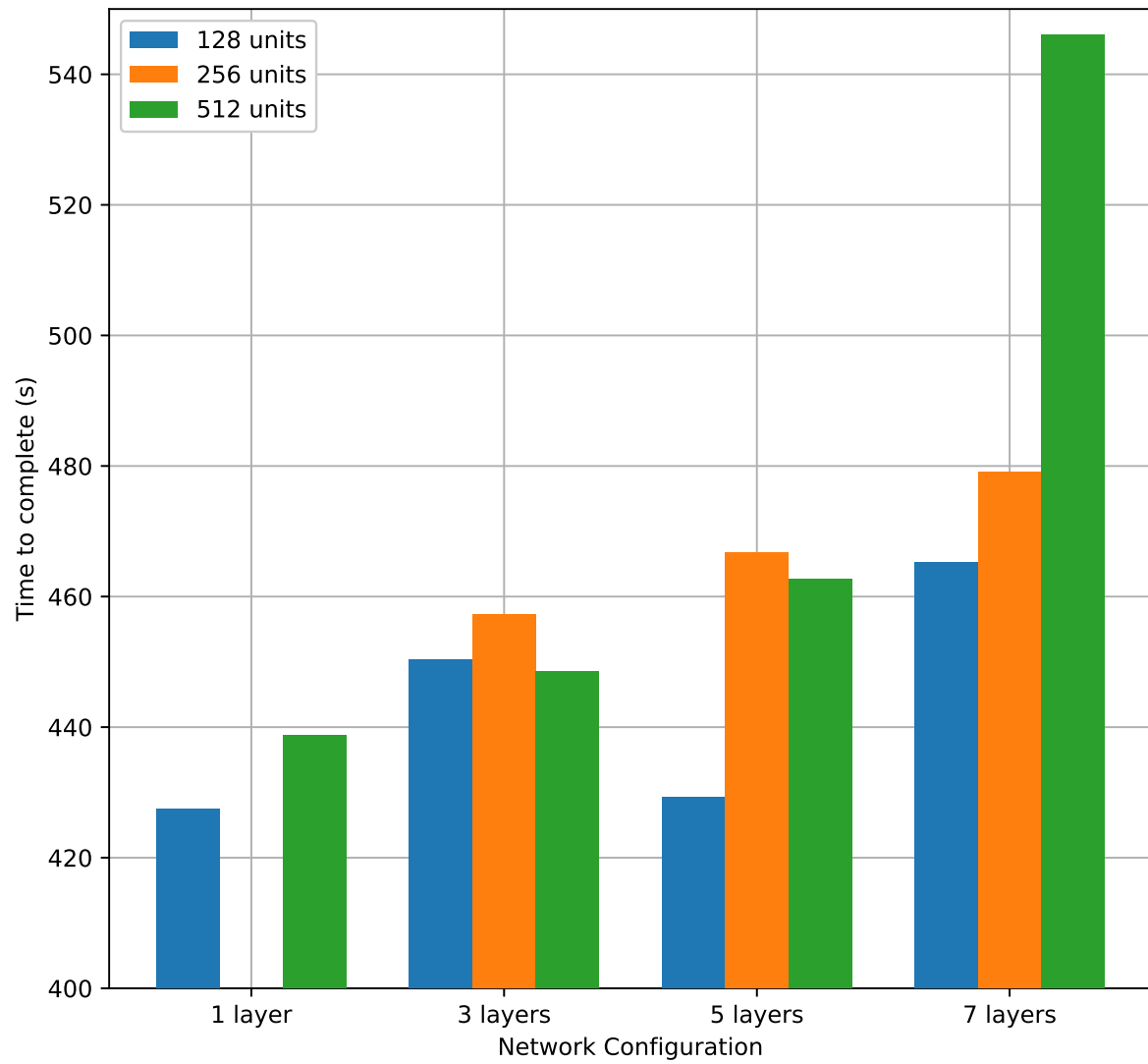


Figure 4.3: Test results for moving course & static brain

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 4.4: Test results moving course, moving brain

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: Final training results

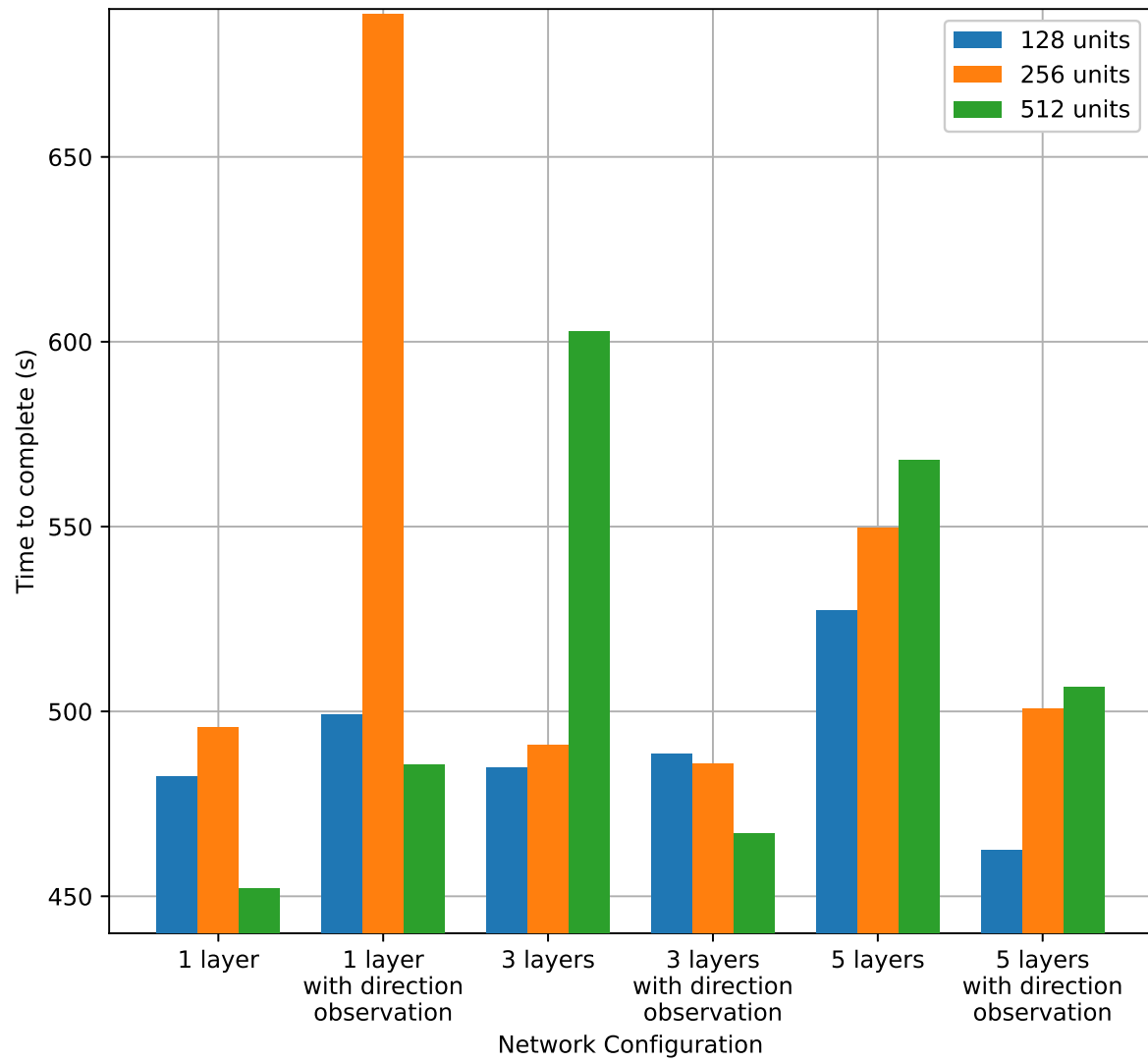


Figure 4.4: Test results for moving course & moving brain

Network Configuration	Bullet Observations	Time to complete (s)
1 layer, 128 units	Bullet's direction 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 direction 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 direction 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 direction 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: Final training results with observation comparasion

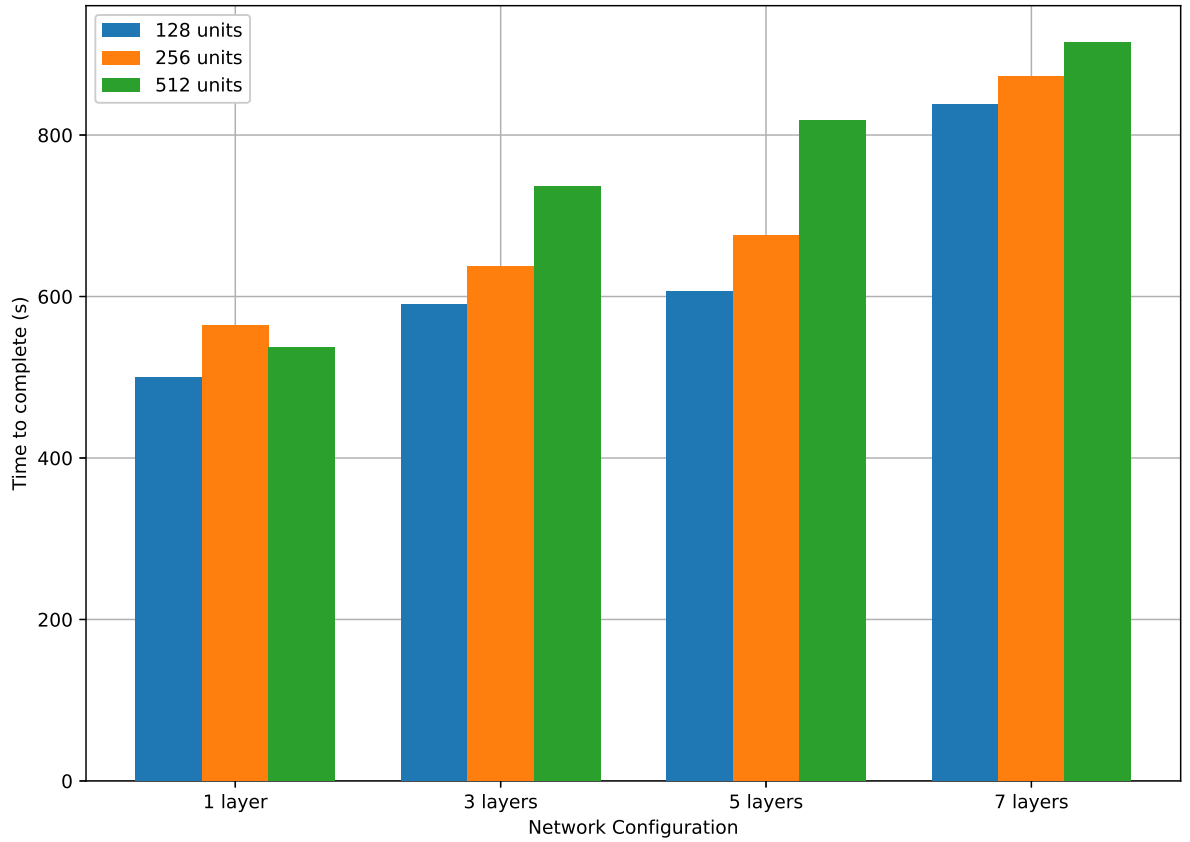


Figure 4.5: Test results for shooting a moving target

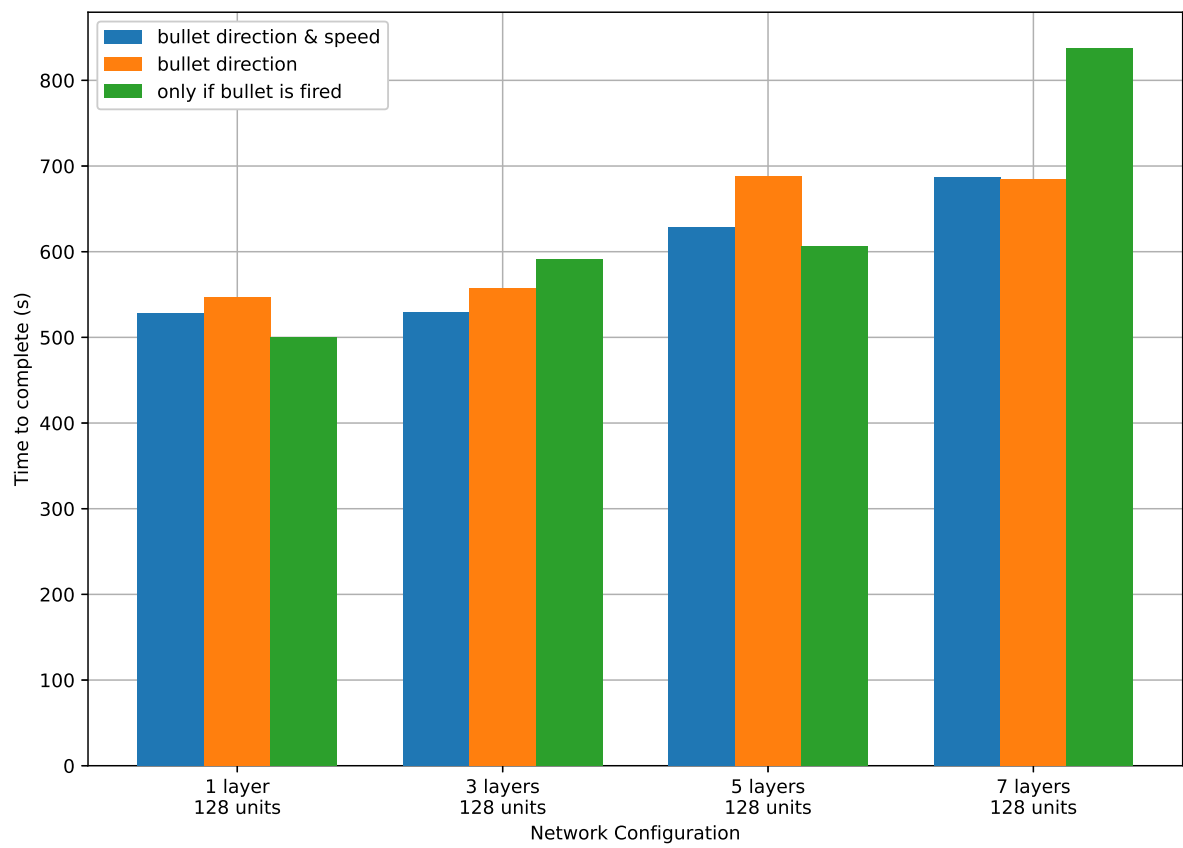


Figure 4.6: Test results for shooting a moving target based on bullet observations

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] Donald E. Knuth. “Structured Programming with *Go to* Statements.” In: *ACM Comput. Surv.* 6.4 (Dec. 1974), pp. 261–301. ISSN: 0360-0300. DOI: [10.1145/356635.356640](#). URL: <https://doi.org/10.1145/356635.356640>.
- [4] 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 \[cs.LG\]](#).
- [5] 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 \[cs.AI\]](#).
- [6] Tim Pearce and Jun Zhu. *Counter-Strike Deathmatch with Large-Scale Behavioural Cloning*. 2021. arXiv: [2104.04258 \[cs.AI\]](#).
- [7] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. *A Game Theoretic Framework for Model Based Reinforcement Learning*. 2021. arXiv: [2004.07804 \[cs.LG\]](#).
- [8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: [1707.06347 \[cs.LG\]](#).
- [9] Yaodong Yang and Jun Wang. *An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective*. 2021. arXiv: [2011.00583 \[cs.MA\]](#).