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

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 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

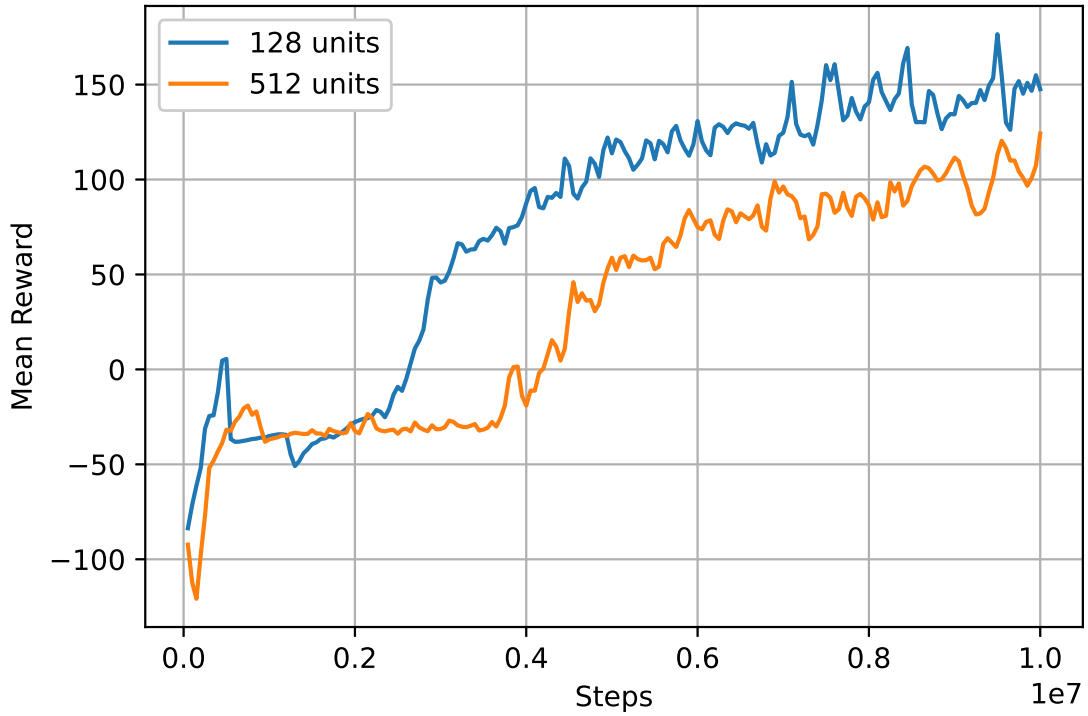

Figure 3.1: Training results with 1 layer of units

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.

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 si de curiosity

TODO: se poate observa ca la 128 units e cel mai bine

Despre ce sa scriu: - ce observatii am ales - ce recompense si pedepse ii dau - faptu ca

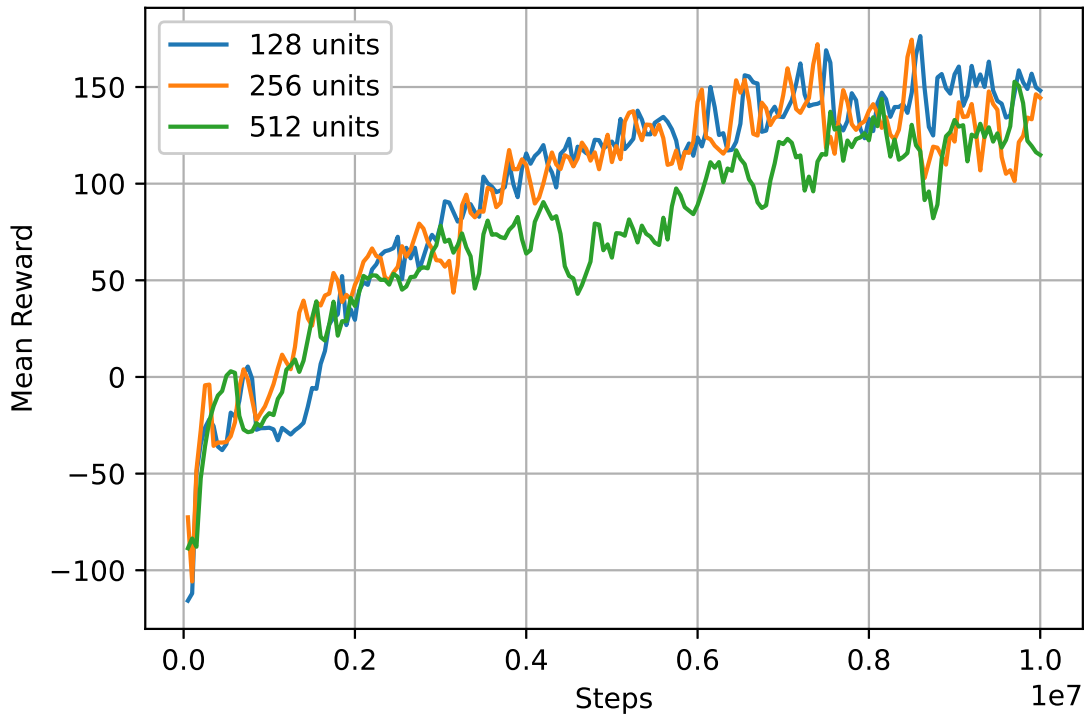

Figure 3.2: Training results with 3 layers of units

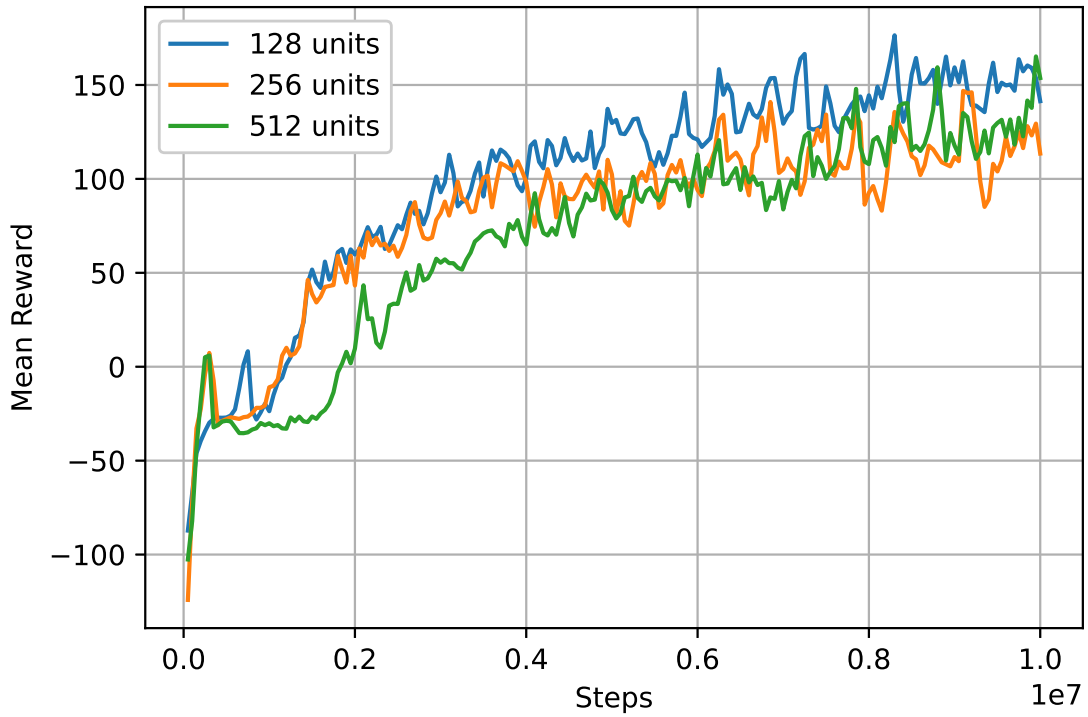

Figure 3.3: Training results with 5 layers of units

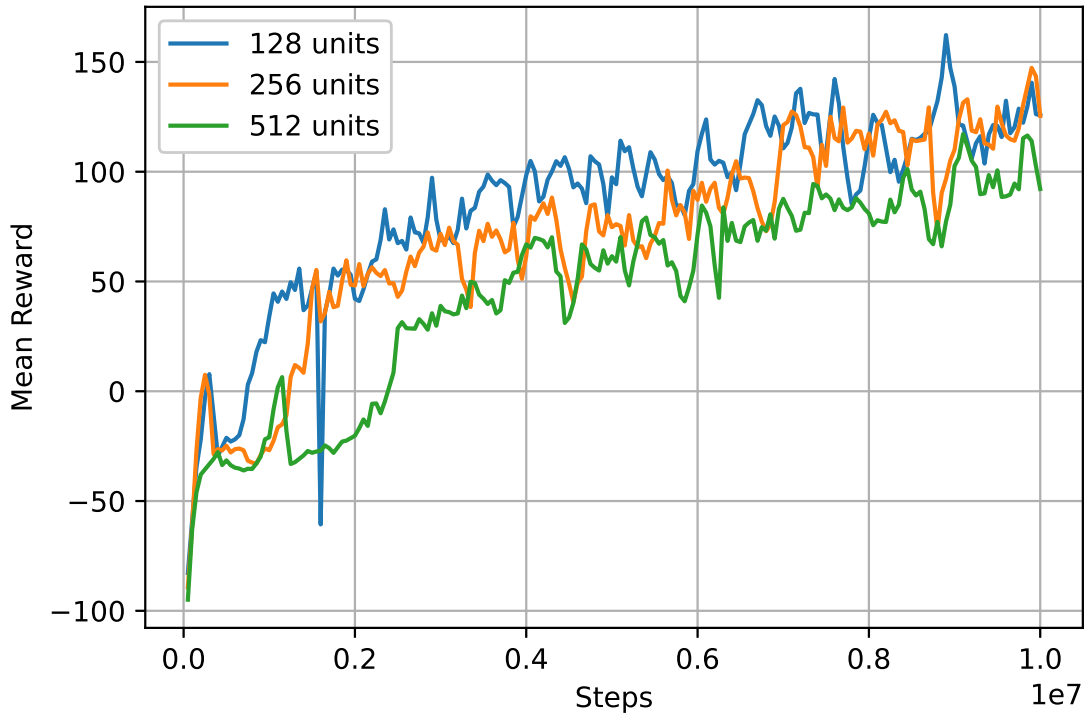

Figure 3.4: Training results with 7 layers of units

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 3.2: Final training results

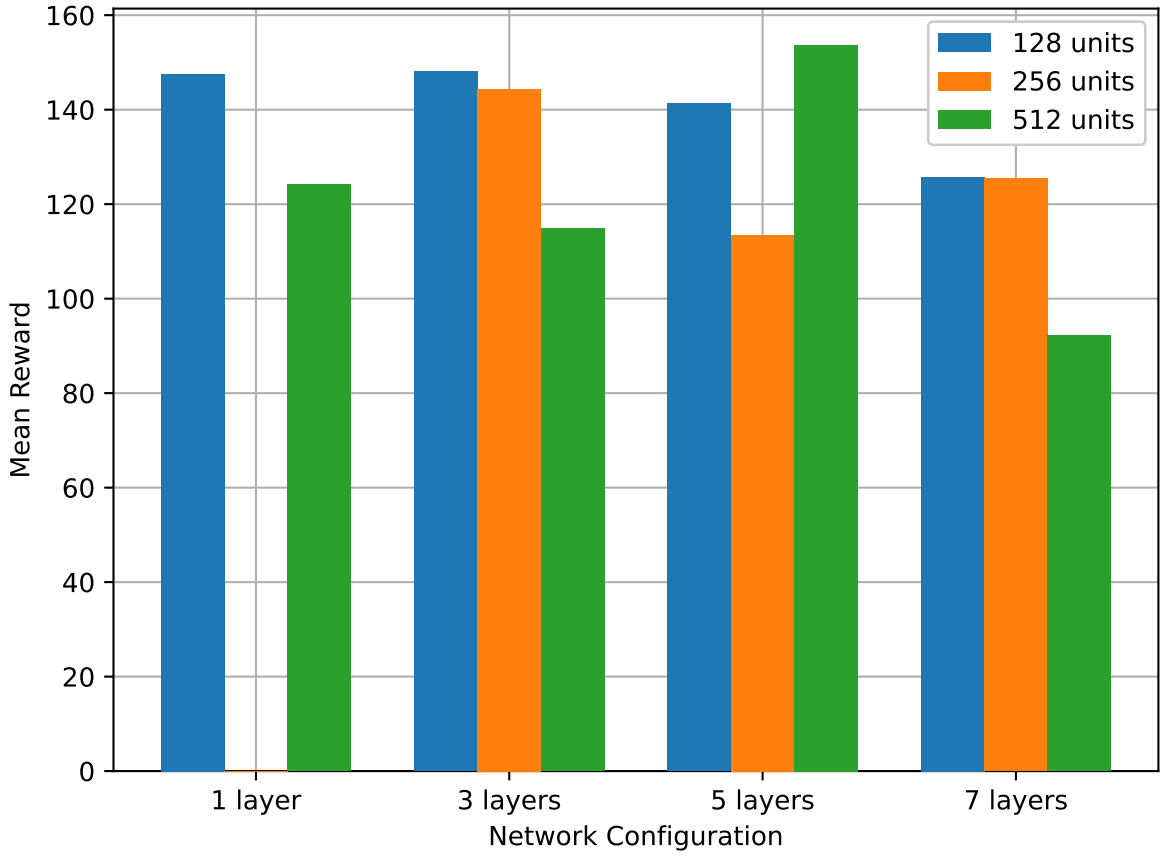

Figure 3.5: Training results for all network configurations

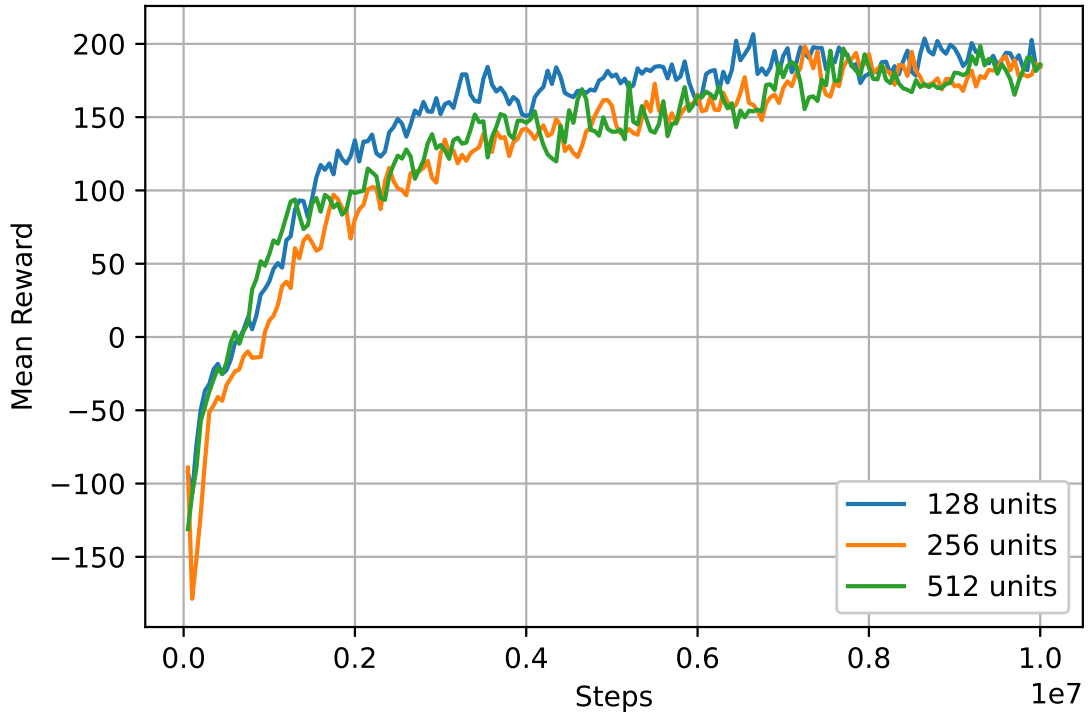

Figure 3.6: Training results with 1 layer of units

initial nu a mers cand am bagat pozitiile - am 30 de agenti care se antreneaza in paralel - la training sa zic despre configuratiile folosite si sa fac grafice cu mean rewards - ca i-am bagat

3.2 Reaching a moving target

TODO: - sa zic ca am incercat sa ii bag memorie recurenta si nu a mers

3.3 Shooting a moving target

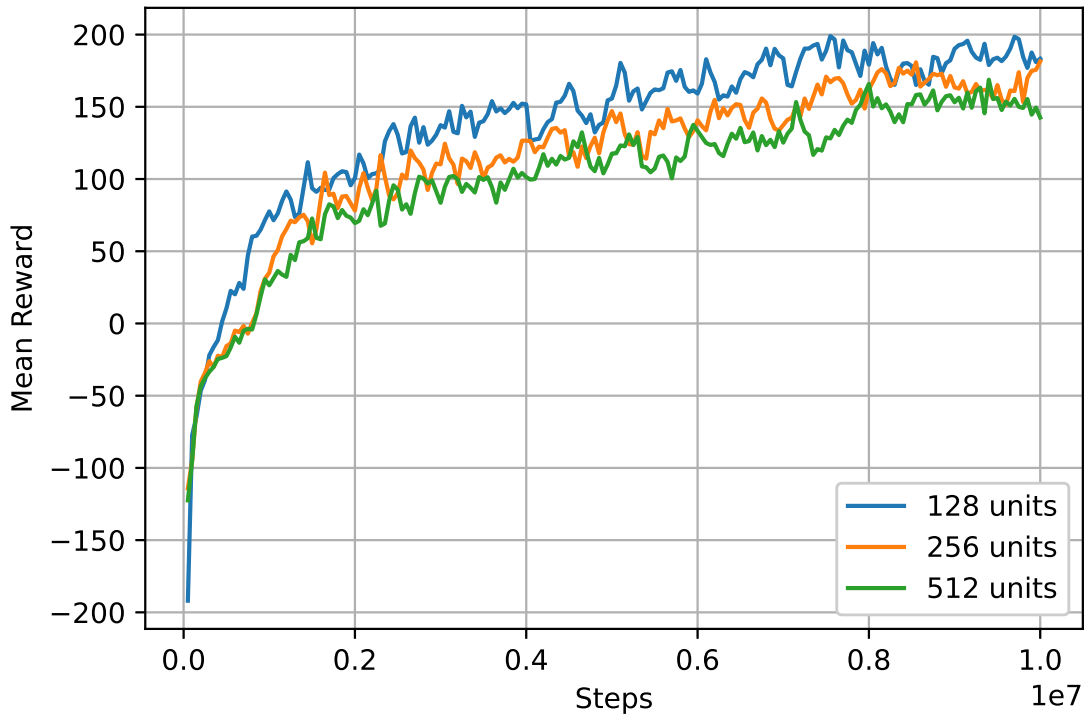

Figure 3.7: Training results with 3 layers of units

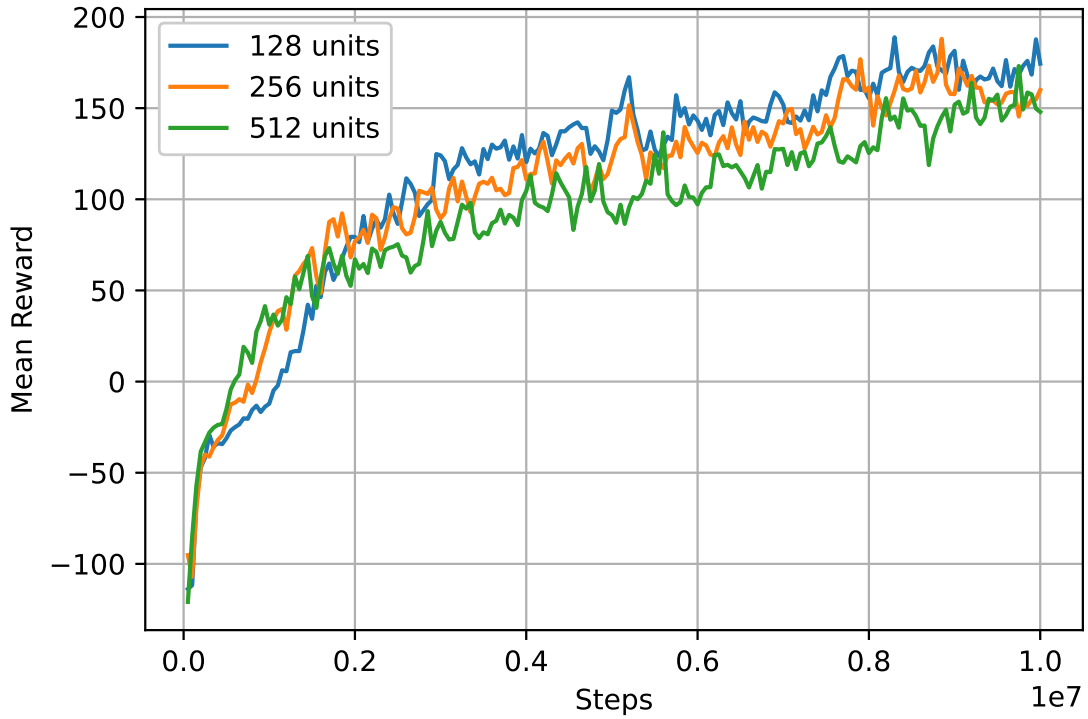

Figure 3.8: Training results with 5 layers of units

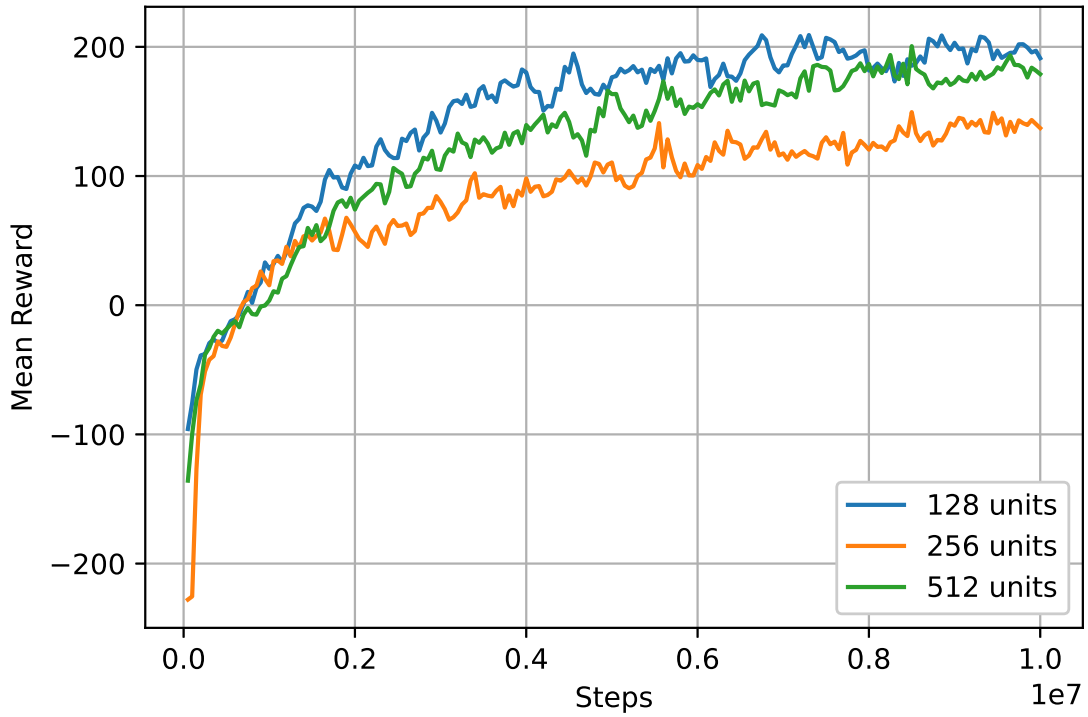

Figure 3.9: Training results with 1 layer of units and with observation of target's direction

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 3.3: Final training results

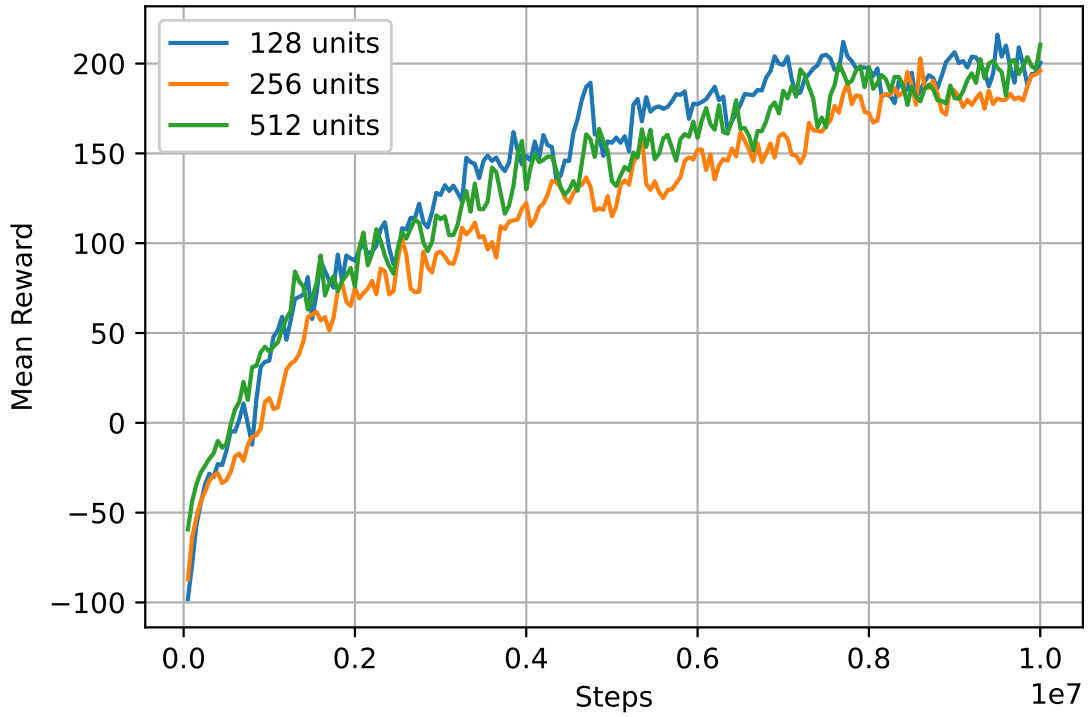

Figure 3.10: Training results with 3 layers of units and with observation of target's direction

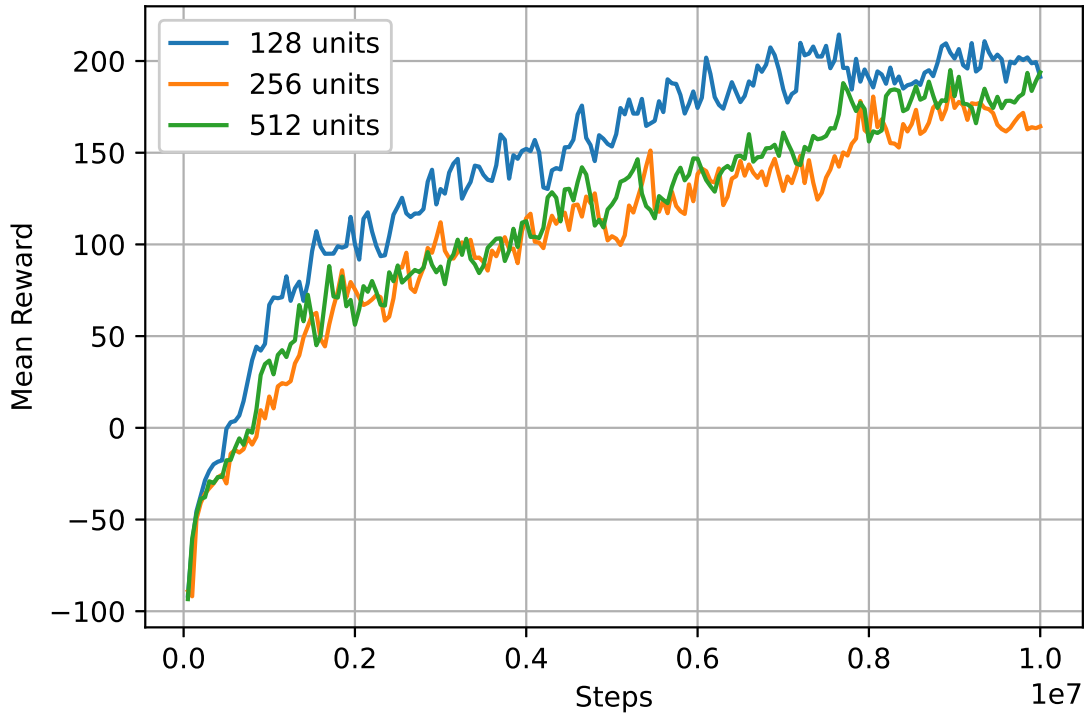

Figure 3.11: Training results with 5 layers of units and with observation of target's direction

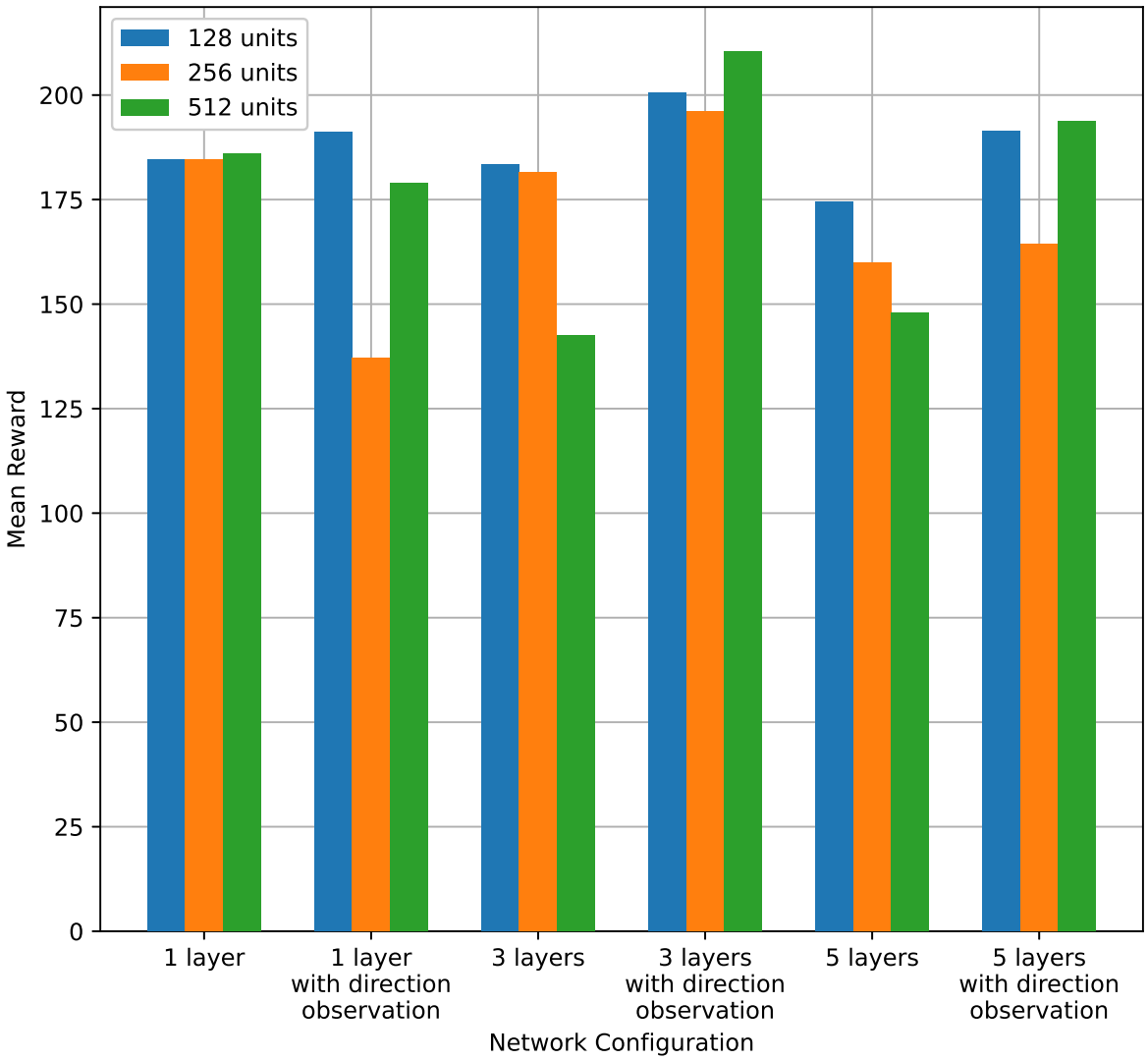

Figure 3.12: Training results for all network configurations

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].