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

TODO: Pedepse si recompense

3.1.3 Training

TODO:

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

3.2 Reaching a moving target

3.3 Shooting a moving target

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].