TÉCNICO LISBOA

Autonomous Agents and Multi-Agent Systems: Project OpenAI - Hide and Seek

Paolo Frazzetto
ist194942

Floris Heijmans
ist197657

Eetu Laine
ist195105

1 INTRODUCTION

Developing artificial intelligence systems that can learn to cope with real-world tasks has been a goal in the research community for a long time. Systems that can interact with the physical world and develop emergent behaviours have been one of the priorities [1]. There have been numerous novel pursuits in the field in recent years, for example in dexterous in-hand manipulation [2, 3] and complex body locomotion [4, 5]. Traditionally, these kinds of systems have been trained either by defining *cost/reward* functions or alternatively by gathering observations and exploiting supervised methods. These approaches, however, often fail to scale to complex real-world settings. The ability to do so usually depends on the extensiveness of the environment and its accuracy of mimicking the real world.

A company that aims to supply such environments is *OpenAI* [6]. OpenAI is a private-founded research laboratory based in San Francisco that, by their own words, aims to ensure that artificial intelligence benefits all of humanity. One of the resources they provide is the so-called *OpenAI Gym*. This gym contains a multitude of environments for which users can create or customise algorithms to reach certain goals. Seeing how creating such environments can take up a lot of time, this resource allows for a increased focus on improving the algorithm aspect.

Although the behaviours that emerge in these systems are specific to the nature of the task, research in the field of Artificial Intelligence can provide some crucial insights to approach open-ended, real-world problems, in a similar fashion of a more familiar course of adaptation: the evolution of human intelligence.

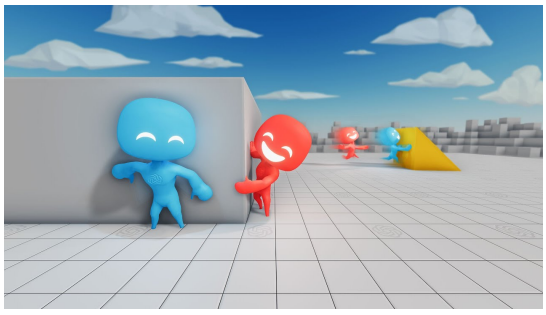

Figure 1: Example of the Hide and Seek environment in the OpenAI - Gym

2 MODELLING THE MULTI-AGENT SYSTEM

2.1 Problem Definition

We suggest to study the real-world problem of *hide and seek*, the popular children's game. In this scenario, there are two teams with distinct tasks - hiding and seeking. The hiders are tasked with avoiding being seen by the seekers, and the seekers conversely with finding and observing the hiders after a predetermined preparation time. The simulation environment contains objects that the agents can grab and fix in place. Furthermore, there are rooms and walls that cannot be moved. The setting differs significantly from the typically studied classical games, such as checkers and backgammon. In this case, the environment and the agents bear more resemblance to the physical world.

This problem holds all the interesting factors that one should look for when simulating a multi-agent environment. Agents in the system can either hide or seek, so the primary aspect of *competition* is present. Additionally, if the system involves multiple hiding and/or seeking agents, the agents can thus *cooperate* in their respective goals. These goals are also quite simple in nature, since it is quite obvious what each agent should do. This helps in keeping the system itself simple, focusing on achieving complex behaviour of agents rather than the rules themselves.

The reference paper's [1] main contribution to the scientific community is the evidence for complex behaviours arising as a result of reinforcement learning. Indeed, even though the AI agents had not received explicit instructions about how to play, they soon learned to run away and chase. After hundreds of millions of games, they learned to manipulate their environment to give themselves an advantage. The hiders, for example, learned to build miniature forts and barricade themselves inside; the seekers, in response, learned how to use ramps to scale the walls and find the hiders. Eventually, as many as six distinct strategies emerged, going well beyond what the human programmers had imagined. Moreover, their framework code has been made available for other researchers to pick up where they left off [7].

2.2 Objectives and Requirements

For this project, we start with simple rule-based policies for the agents, incrementally making the policies more sophisticated by incorporating different learning and planning techniques in the domain of Policy Optimisation, Reinforcement Learning and *deep RL*. At each iteration of the process, we measure the performance of the agents and summarise the results that each method produces.

The simulation setting introduced by [1] will be used. This setting uses the OpenAI - Gym resource and more specifically, the Hide and Seek environment. Having this prepared environment, allows us to focus purely on creating, tweaking and analysing algorithms for our multi-agent system, without having to build everything from scratch.

Our ideal goal would be to reproduce at least the basic results of the OpenAI team, considering our limited computing resources. In other words, we aim to *reach some emergent behaviour learned by the agents*, evolving organically and not as a result of pre-built mechanisms.

3 SYSTEM CHARACTERISTICS

3.1 Fundamental Principles

Agents are tasked with competing in a two-team hide-and-seek game in a physics-based environment. The hiders conceal themselves in a set environment, to be found by one or more seekers. There are objects scattered throughout the environment that the agents can grab and also lock in place, along with randomly generated immovable rooms and walls that the agents must learn to navigate. Before the game of hide-and-seek begins, the hiders are given a preparation phase where the seekers are immobilised, giving the hiders a chance to run away or change their environment.

Agents are given a team based reward: hiders are given a reward of 1 per time step if all hiders are hidden and -1 if any hider is seen by a seeker. Seekers are given the opposite reward, -1 per time step if all hiders are hidden and +1 otherwise.

3.2 Agent Design

3.2.1 Sensors. The agents sense the location and velocity vectors of the objects and agents in the environment, as well as their respective sizes. If objects do not lie within a 135 degree cone centred around the line-of-sight of the agent, they are hidden from it. Furthermore, the agents are equipped with range sensors evenly distributed around them, in similar fashion to a radar, so that they can somewhat perceive and escape from a chase outside their field of vision. The agents can also observe their own team, the team of other agents and the remaining preparation time.

3.2.2 Actuators. The agents are modelled as spherical objects. They have three types of actions available to them: *moving* along the x- and y-axes, and rotating around the z-axis. Agents can also *grab* and *lock* objects. Objects can be unlocked by friendly agents. For an agent to reach an object, it must be within a sufficient distance from the object.

3.3 Environment

In the setting of [1], the simulation consists of 1-3 hiders and 1-3 seekers and 3-9 movable boxes of various shapes. The players are confined in an arena to prevent the hiders to just endless run as far away as possible. Additionally, there are the previously mentioned rooms with walls which the agents cannot manipulate. Finally, the simulation is implemented utilising the physics engine MuJoCo [8].

4 ENGINEERING THE AGENTS' SYSTEM

In order to develop our agents' system, three approaches described in the following subsections were undertaken. All the code and collected data are available at the following GitHub repository and related branches [9].

The environment and the agents within can be controlled by means of the newly designed class `RuleAgent.py` that functions as a base for the modified `policy_viewer.py`, that runs a policy and optionally displays it. The chosen environment is `hide_and_seek_quadrant` taken from the examples. It consists of 2 hiders and 2 seekers, 2 boxes and 1 ramp randomly placed on a square playground, in which a quarter is surrounded by a wall with two entrances (see Figure 2). These entrances are also positioned and sized randomly along the walls, with one entrance per side of the quadrant and the entrance sizes large enough to fit through.

Actions are characterised in the following not-so-intuitive but practical fashion: each of the three actions takes integer values from 0 to 10 (included), where 0 means maximum speed for low/-left/counterclockwise rotation, 5 stay still, 10 is maximum speed in opposite direction, i.e. up/right/clockwise.

The agents position and velocity, together with the objects placement, are initialised randomly at every game. Each run consists of 240 time steps.

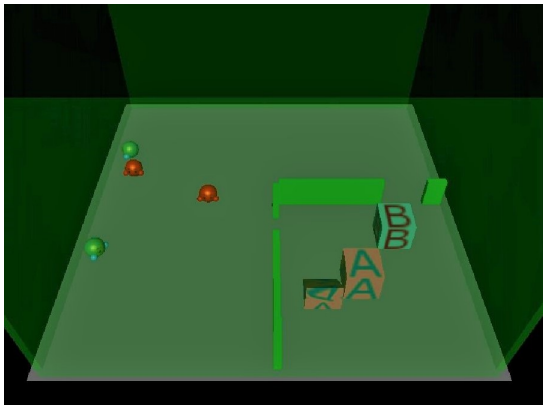

Figure 2: Environment of interest, with 2 hiders (green), 2 seekers (red), 2 randomly placed boxes and 1 ramp.

4.1 Random Movements

First of all, as a baseline and further reference agents are let move at random. At each action is therefore assigned an uniformly distributed value in $[0, 10]$ for every time step. Recall then that the rewards are only updated whenever an agent of the opposing team enters in the field of view.

4.2 Rule Based

The second proposed approach is a simple rule-based policy that work as follows: a seeking agent moves randomly until it sees a hider, then it starts to pursue it moving in the same direction. The hiders, conversely, move randomly until they are being seen and then they start running away. Thus, the agents follow a purely reactive architecture.

4.3 Deep Q Learning

To go beyond the rule-based approach, Q-learning was contemplated at first but a major limitation is that it only works in environments with discrete and finite state and action spaces. One solution for extending Q-learning to richer environments, like the one considered, is to apply function approximators to learn the Q-value function, taking states as inputs, instead of storing the full state-action table. Since deep neural networks are powerful function approximators, in recent years they have been adapted for this role [10], establishing the so called *Deep Q Learning*.

In DQL, preprocessed game states are given to a neural network, which will return the Q-values of all possible actions in the state. The next action is selected using an ϵ-greedy policy with probability ϵ. An action is chosen and with probability $1 - \epsilon$ of the actions that

have a maximum Q-value, i.e. $a = argmax(Q(s, a))$. Then each agent performs this action in state s and moves to a new state s' to receive a reward (that is the exact opposite of the other agent team). After preprocessing, the state s' is the input of the next game step. These transitions are stored in a replay buffer as $\langle s, a, r, s' \rangle$. After a certain amount of runs, some random batches from the replay buffer are sampled and used to calculate the loss and update the network weight with backpropagation in order to minimise this loss. The loss function is the squared difference between target Q and predicted Q. This process is repeated until the user decides to stop it (e.g. when the agents have achieved a desired loss).

Notice that, contrary to the reference paper, this approach can be categorised as *online* learning; no data samples are stored for each training epoch, meaning that this process can virtually go on (and improve) indefinitely.

The above DQL network can be implemented by means of the `TF-Agents` library [11], which was built on top of TensorFlow and OpenAi Gym, so it would be possible to be pulled off with the existing settings. To mimic the original architecture, first of all recall that each agent perceives its surroundings by means of a *lidar*, that returns the sensed distances around it. This input has been processed by means of a 1D circular convolution layer, so that the network sees this data as circular. Note, that the authors of [1] also do this. Then, all the agents and objects statuses are connected to form the input layer that is fed to the DQL network. Additionally, the agent's actual field of view is used to mask the agents and objects outside of it. In this way, unseen items are not used for prediction. Finally, by nature of the library, the output of the learning agent is a scalar integer value. Through a straightforward mapping this singular value can represent all three movement values.

4.4 Data Description

At every step of the game, the environment provides the following relevant quantities that are being used to describe, tune and ultimately run the agents. An example file is available on the GitHub repository. `observation_self` contains the position and speed for each agent, whereas `agent_qpos_qvel` gives those information to the other agents, that are masked according `mask_aa_obs`. Analogous data structures are present for the boxes and ramp. Finally, the objects pre-fixed with `action` obviously define the actions for each agent. These observations and rewards were both used to better understand the model, and then to control the agents and gather measurements.

5 ANALYSIS OF THE RESULTS

A dedicated class `Measure` was employed to collect meaningful data used to analyse and compare the results. Besides the final reward, it has been considered to measure as well the cumulative movements of the teams and number of times an object was moved. See Table 1 for an example. In order to have substantial statistics, about 500 games were run for each scenario.

5.1 Random vs. Rule-based

From looking at the basic descriptive statistics (Tables 3 and 4 in appendix), some things can be suspected already. Firstly, hiders seem to move more than seekers in both agents; recall that indeed they have extra preparation time in which the seekers are not allowed to move. Random agents also seem to move around more than rule-based agents, due to the intrinsic unconstrained nature. Lastly, in the rule-based scenario average rewards significantly improve for the seekers and agents avoid to bump into objects more, meaning that this policy, although naive, is already a relevant improvement compared to the random setting. The goal of mimicking a real world scenario where seekers actively try to *seek* hiders has thus been achieved. Due to naivety of the agent, the hiders have not been able to show behaviour that improves their chances, such as moving into the quadrant or using objects to hide. Moving away or out of sight of seekers seems to have been a relatively less impactful improvement than the chasing rule of the seekers.

	objects moved	hiders movement	seekers movement	hiders reward	seekers reward
0	11	35.37	22.77	-46.0	46.0
1	0	35.25	22.21	-6.0	6.0
2	18	35.77	20.51	-34.0	34.0
3	81	33.69	24.80	48.0	-48.0
4	81	35.45	19.69	0.0	0.0

Table 1: Example of five rows of the performance measures that were obtained for the rule-based agent.

	statistic	p-value
objects moved	39846.5	1.07e-06
hiders movement	44.0	2.40e-83
seekers movement	1649.0	3.32e-79
hiders reward	19816.5	4.07e-36
seekers reward	19816.5	4.07e-36

Table 2: Results of the Wilcoxon tests done on the differences between all the measures of random and rule-based agents. All tests are significant with $\alpha = 0$.

To test statistical difference, the Wilcoxon signed-rank test has been used. It tests the null hypothesis that two related paired samples come from the same distribution. It is a non-parametric version of the paired T-test that does not assume normally distributed samples. The results are reported in Table 2 showing that indeed the two behaviours are statistically different.

Figure 3 displays the correlations among each paired quantities for the rule-based agents, pointing out a minor correlation between seekers' movements and rewards. This can be translated to the fact that the seekers spend significant time chasing the hiders and consequently improving their rewards.

Having a deeper look at the measurements distributions (Figure 4 in appendix), it can be noticed once more that randomly moving agents bump more into objects. The outliers in the rule-based agents are probably cases where both hiders spawn inside the room. Additionally, there seem to be more outliers for the rewards. These can be explained by cases where both hiders are spawned inside the quadrant. In contrast to random agents, they are more inclined to stay inside the quadrant.

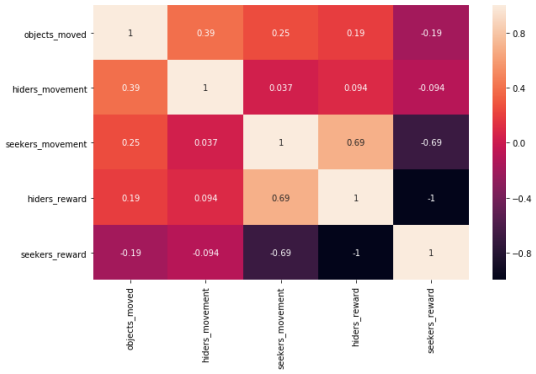

Figure 3: Correlations for rule-based measurements.

5.2 Deep Q Learning

Unfortunately, the technical implementation turned out to be significantly more complex than initially thought; even though building the DQL network was feasible, translating the input states into a format suited for interfacing TF-Agents was the major issue that could not be solved due to time constraints. Building a neural network from scratch or using other libraries was not a valid alternative either, due to already having started going down the path of the DQL agent. We do however believe that, given a correct translation of the environment format of I/O to the TF-Agents format, it would be possible to achieve online training for this environment.

6 FURTHER DEVELOPMENTS & CONCLUSIONS

In this work we had the chance to deal with a state-of-the-art framework in the field of Reinforcement Learning (with the library used not even being a year old [11]), applied on a real-word multiagent scenario. Even though the code is open sourced, there is no real documentation available (outside of a single tutorial that was for the most part irrelevant to our case), so the process of setting up the environment and understanding its inner workings ended up being more complex than expected. The time spent to achieve what was originally proposed therefore increased by quite a lot. Nevertheless, our rule-based approach, in its simplicity, proved to be a basic achievement that can be easily improved. For instance, agents could be implemented according to a hybrid fashion, that beyond reactively running and chasing, would contemplate some planning. They could try to get out of the line of sight of agents who see them or hiders could find a point that's far away from seekers and go there. Other options would be creating features that check whether the hider is inside the quadrant, and if so, staying put or actively looking for the doors of the quadrant to try and enter (which could of course also be used by the seekers to find the hiders).

Anyway, our main focus was to implement a DQL network, hoping to go beyond the classical architectures. This task revealed to be more challenging than anticipated, on top of the already limited time and computing power. Although this endeavour was unsuccessful, its execution has tested our knowledge, it has arisen some interesting discussions among us and it has let us gain new valuable experience in the fascinating field of AI.

REFERENCES

[1] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[2] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[3] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[4] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[5] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL http://arxiv.org/abs/1606.01540. cite arxiv:1606.01540.

[7] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula - github repo. https://github.com/openai/multi-agent-emergence-environments, 2019.

[8] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[9] Eetu Laine Paolo Frazzetto, Floris Heijmans. Aamas project 2020. https://github.com/EetuaLaine/MultiAgentSystems20, 2020.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Oscar Ramirez Pablo Castro Ethan Holly Sam Fishman Ke Wang Ekaterina Gonina Neal Wu Efi Kokiopoulou Luciano Sbaiz Jamie Smith Gábor Bartók Jesse Berent Chris Harris Vincent Vanhoucke Eugene Brevdo Sergio Guadarrama, Anoop Korattikara. Tf-agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018.

APPENDIX

Random Agent					
	objects_moved	hiders_movement	seekers_movement	hiders_reward	seekers_reward
count	499.000000	499.000000	499.000000	499.000000	499.000000
mean	34.236473	34.351547	20.560247	-13.070140	13.070140
std	28.567756	2.217739	1.651647	22.508211	22.508211
min	0.000000	27.718158	15.977831	-48.000000	-48.000000
25%	11.000000	32.668315	19.371604	-30.000000	-2.000000
50%	30.000000	34.370001	20.627567	-18.000000	18.000000
75%	52.000000	35.841089	21.667857	2.000000	30.000000
max	140.000000	40.832329	25.224493	48.000000	48.000000

Table 3: Statistics of colleccted data for baseline Random Agents.

Rule Based Agent					
	objects_moved	hiders_movement	seekers_movement	hiders_reward	seekers_reward
count	499.000000	499.000000	499.000000	499.000000	499.000000
mean	26.162325	17.493710	13.220787	-32.080160	32.080160
std	28.387163	7.000393	4.181318	20.807629	20.807629
min	0.000000	3.481080	2.084833	-48.000000	-48.000000
25%	0.000000	12.062239	10.275816	-48.000000	26.000000
50%	19.000000	17.489332	13.121015	-40.000000	40.000000
75%	45.000000	22.408089	16.365567	-26.000000	48.000000
max	158.000000	36.319947	24.429138	48.000000	48.000000

Table 4: Statistics of colleccted data for Rule Based Agents.

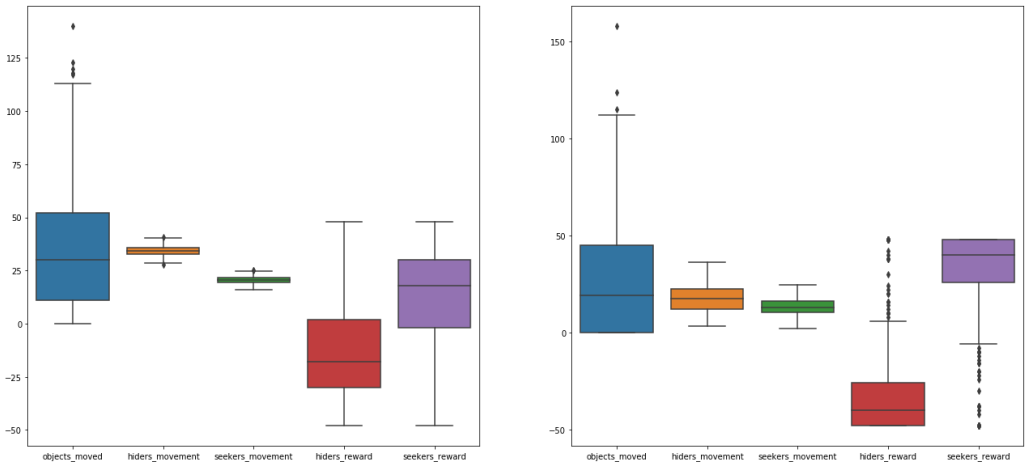

Figure 4: Distributions for random (left) and rule-based (right) measurements.