Autonomous Agent in a Cooperative Game Environment

Alexandre Rodrigues - 89404
alexandre.rodrigues@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisboa, Portugal

Tomás Zaki - 79690
tomas.zaki@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisboa, Portugal

Mikko Seppi - 104418
mikko.seppi@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisboa, Portugal

ABSTRACT

This project's goal is to compare and study several approaches to competition of agents. Two agents, whose architecture will vary, will have to compete in order to complete a series of puzzles. By measuring performance indexes of each approach, we will then conclude which of these is more suited for the problem posed.

KEYWORDS

Autonomous Agents, Q-Learning, DQN

1 INTRODUCTION

For the purpose of our studies on Autonomous Agents and Multi-Agent Systems, our proposed project is one where autonomous agents, with different architectures, must complete a series of puzzles, by interacting with their environment in order to reach the finish line. Each agent can only see their environment and can only interact with objects that are placed in front of them. The tasks to be performed will be described in the next section. After each run, a set of metrics will be taken with the intent of performing a comparative analysis of the agent architectures that will be implemented during the project's development. Ultimately, the goal will be to reduce the time and steps of the agents need to complete the puzzles and reach the finish line.

2 PROPOSED APPROACH AND METHODOLOGIES

2.1 Environment

All the approaches will be tested on the same Environment. The project's environment is a two sides vertical runner with three obstacles. The length of the runners will be determined by the number of obstacles and the width will have a fixed value.

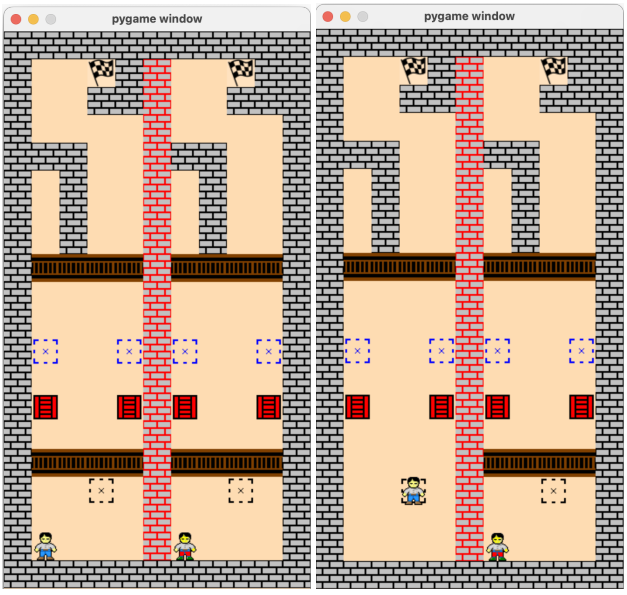
There are several types of puzzles that agents have to solve in order to reach the final goal(complete all puzzles) in the environment:

The trials consist of:

- Find and station in a dock to move the obstacle
- Push boxes to their respective locations to move the obstacle
- Find and touch the finish flag

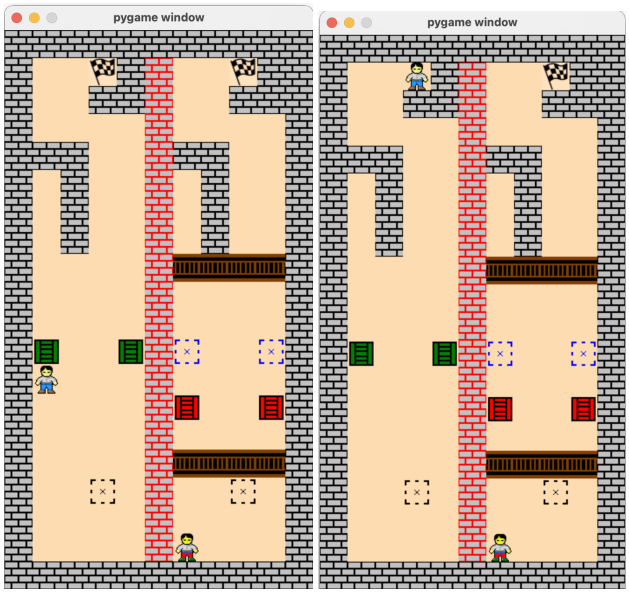
Given all this we characterize the environment as follow:

- Inaccessible
- Deterministic
- Static
- Discrete
- Episodic



(a) Starting agent position. (b) Agent completed puzzle 1.

Figure 1: Agent evolution in the environment



(a) Agent completed puzzle 2. (b) Agent completed puzzle 3.

Figure 2: Agent evolution in the environment

2.2 Autonomous Agent System

In our Autonomous Agent system, agents will try to complete the previously mentioned challenges as fast as possible. Each agent has the following actuators:

- Move up, down, left or right
- Stand in buttons to press them
- Push Boxes

Each agent has the following sensors:

- Visual
- Memory

2.3 Architecture

For this project we considered the following agent Architectures:

- Random: Here the agent chooses his action at random. We expect this to be the worse performing architecture as it does not learn between episodes.
- Q-Learning with e-greedy policy
- Deep Q Network with e-greedy policy

To start with, we will explain what we mean by e-greedy policy. The agent has a value epsilon that will determine whether the agent Explores or Exploits.

Result: Selected action
Function *SELECT-ACTION(Q, S, ϵ)* **is**
    n $\leftarrow$ uniform random number between 0 and 1;
    if $n < \epsilon$ **then**
        A $\leftarrow$ random action from the action space;
    else
        A $\leftarrow$ maxQ(S,.);
    end
    return selected action A;
end

Figure 3: Epsilon-Greedy Action Selection

When the Agent Explores, it allows to improve on which action will be better for the current state, so that it learns what to do in future episodes. When the Agent Exploits, it uses the prior knowledge to make decisions on what it perceives to be the better action given the current state.

2.4 Rewards

For puzzle 1 and 3 states are position of the agent and agent gets reward -0.1 for every step and reward 10 when it reach the goal. For puzzle 2 states are position of the agent and position of the boxes. and agent gets reward -0.1 for every step ,0.5 when box is pushed up, -1 when box is pushed down, 5 when box reach the dock and 10 when both boxes in the dock.

2.5 Q-learning

Q-Learning with e-greedy policy is an algorithm that seeks to find the best action to take given the current state. It learns the value of an action in an particular state.

Result: A Q-table containing Q(S,A) pairs defining estimated optimal policy π^*
`/* Initialization */`
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) $\leftarrow$ 0;
`/* For each step in each episode, we calculate the Q-value and update the Q-table */`
for *each episode* **do**
    `/* Initialize state S, usually by resetting the environment */`
    Initialize state S;
    for *each step in episode* **do**
        do
            `/* Choose action A from S using epsilon-greedy policy derived from Q */`
            A $\leftarrow$ SELECT-ACTION(Q, S, ϵ);
            Take action A, then observe reward R and next state S';
            Q(S, A) $\leftarrow$ Q(S, A) + α [R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
            S $\leftarrow$ S';
        while *S is not terminal*;
    end
end

Figure 4: Epsilon-Greedy Q-Learning Algorithm

Q-Learning with e-greedy policy has an agent, a set of states (**s** $\in$ **S**) a set of actions per state (**a** $\in$ **A**). An action transits from state to state and in each state it receives a reward **r** (numeric value) that is also used to measure success or failure. The main goal of the agent is to maximize the total of the rewards. He achieves this by adding as many futures rewards as possible to the current state. The discount factor () how long in the future the agent looks for rewards. For example, if the discount factor value is closer to zero, the agent does not consider rewards in the distant future and if is closer to one he agent is considering rewards in the future.

The Q values are updated using a greedy policy that allows the agent to select the action that most of the time is the optimal action. With probability 1 the agent choose the optimal action according to Q. With probability different than 1 the agent choose a random action.

The agent observes the environment, takes an action to interact with the environment and receives a reward.

2.6 SARSA

State-Action-Reward-State-Action (SARSA) is an algorithm for learning Markov Decision Processes, similar to Q-Learning. The difference between both algorithms is in their approach to policy. SARSA is considered an on-policy technique, where the agent learns the value function according to the policy currently being used, as opposed to Q-Learning, being an off-policy technique.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Figure 5: SARSA Algorithm

2.7 Deep Q-Network

Deep Q-Network is an algorithm developed by DeepMind in 2015 and uses a combination of Q-Learning and Deep Neural Networks. For this algorithm, we were inspired by the Pytorch implementation to train an agent to play a Mario Game. Deep Neural Networks are Artificial Networks, with multiple layers between the inputs and outputs. The idea consists of simulating human brains with neurons, synapses, weights, biases, and functions.

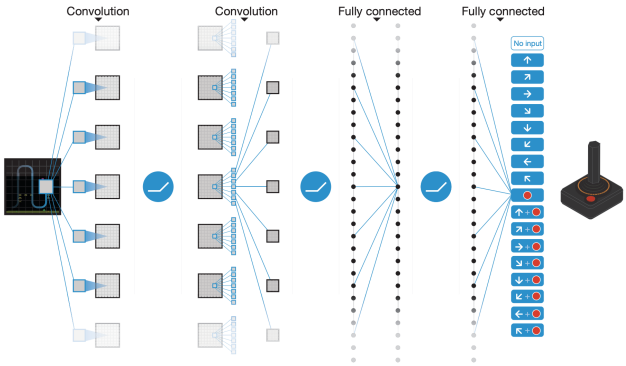


Figure 6: Deep Q-Network Convolutional Network

3 EVALUATION

The evaluation of the Autonomous Agent system will be done using the following set of metrics:

- Number of steps taken by agents
- Value of Reward with time

Our objective will be to compare various architectures throughout 150 episodes as this will give us a better outlook on how each algorithm learns. As mentioned previously, the random architecture is expected to be the worse performing as it does not learn between episodes. When comparing a Q-Learning vs Random, we predict that the Q-Learning Agent will outperform the Random Agent because the latter does not learn from experience. Between Q-Learning and SARSA, we expect that both will have similar behaviours in terms of learning curve, as the implementation is similar. Unfortunately, due to problems with the implementation, we could not do a comparative evaluation of Q-Learning and Deep Q-Network Agent. However we would presume that the DQN agent, due to how it learns, would be a more efficient learner than Q-Learning and therefore, would outperform in the long run. Since all agents have an e-greedy policy, it is in our interest to see how the value of epsilon affects the agent behaviour and how the values of the reward vary with this change as well.

4 RESULTS

Presented in this section are the plots that illustrate the comparison in performance between the Random Agent (Agent 1) and Q-Learning Agent (Agent 2) as a baseline and between SARSA (Agent 1) and Q-Learning (Agent 2).

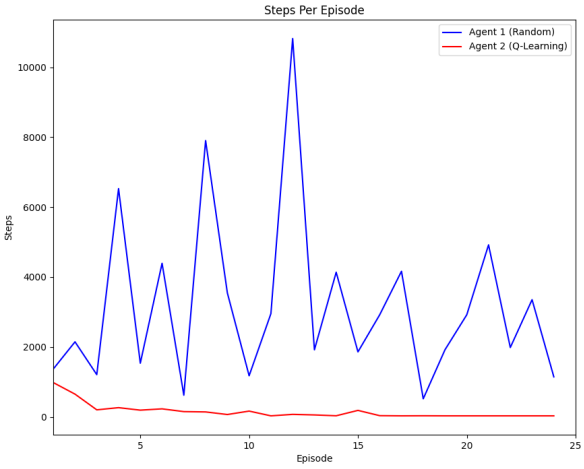


Figure 7: Steps per episode

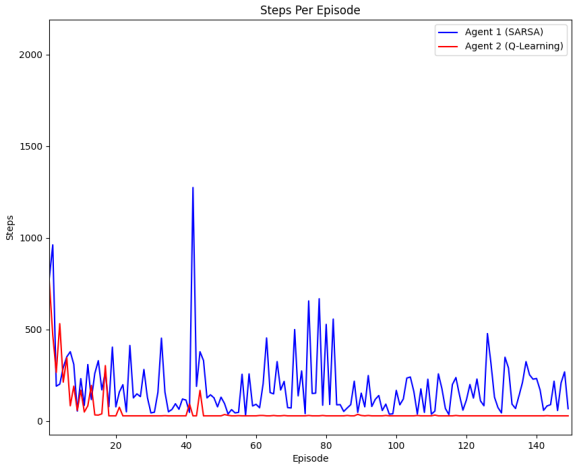


Figure 8: Steps per episode

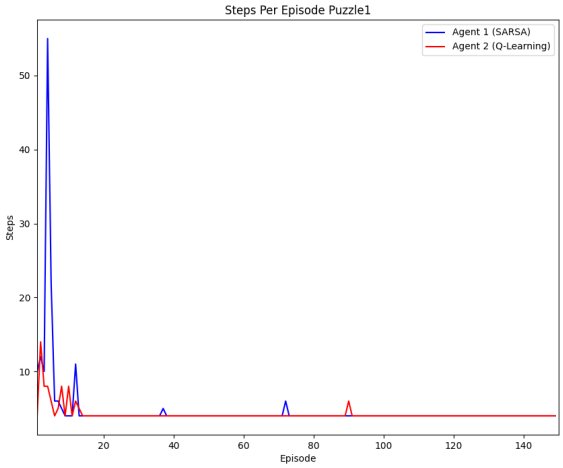


Figure 9: Steps per episode puzzle 1

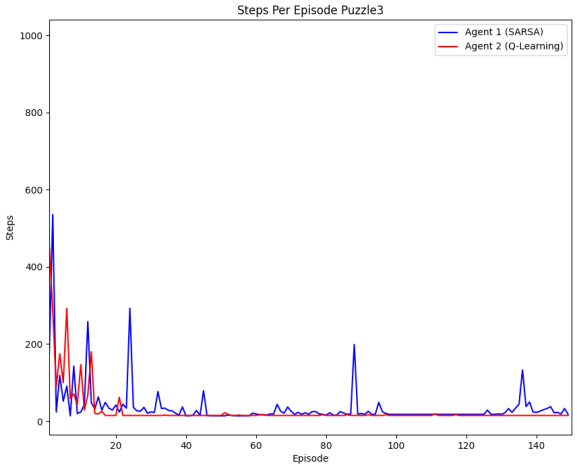


Figure 11: Steps per episode puzzle 3

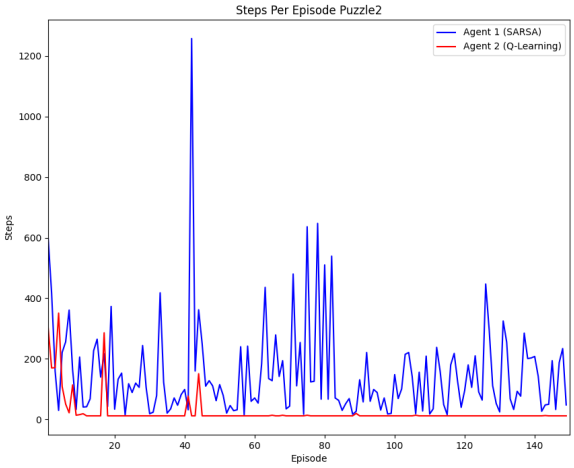


Figure 10: Steps per episode puzzle 2

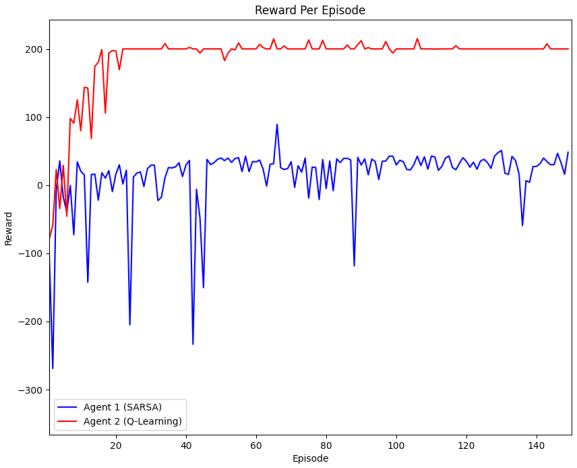


Figure 12: Reward per episode

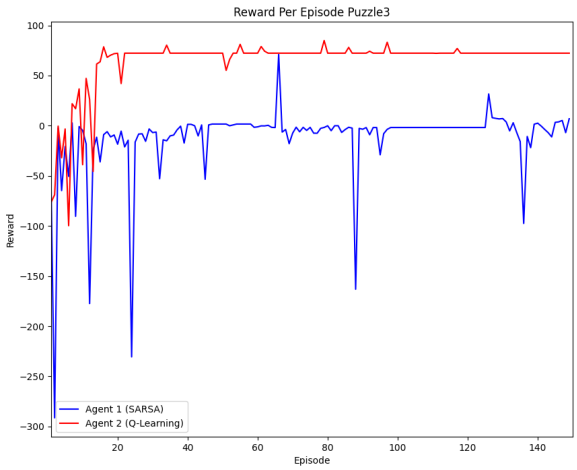


Figure 13: Reward per episode puzzle 1

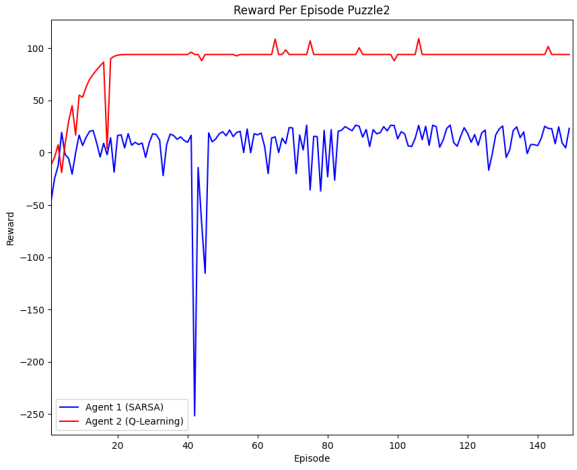


Figure 14: Reward per episode puzzle 2

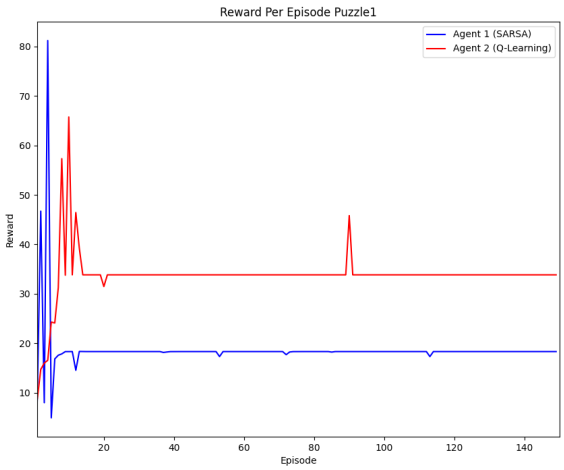


Figure 15: Reward per episode puzzle 3

5 ANALYSIS

As mentioned in the Evaluation section of this report, our agents went through 150 iterations of the game. From the plots presented above, we note that, while the Random agent occasionally outperforms the Q-Learning Agent, we can attribute this to chance. When comparing Q-Learning and SARSA, we note that their learning curves are very similar, by comparing the number of steps and the rewards for each episode between both agents. We see that the hardest task to perform and learn is the box pushing puzzle, because of the bigger quantity of states, and therefore bigger Q-table, that this puzzle requires.

6 CONCLUSIONS

As predicted in the evaluation section and confirmed by our results, the Q-Learning Agent outperforms the Random Agent. Another confirmation obtained by our results is that Q-Learning and SARSA have a very similar performance, with Q-Learning outperforming SARSA sometimes, but both algorithms end up converging. For future works, it would be interesting to compare a bigger variety of Agents architectures, such as DQN and Deliberative Agents for instance. Originally, this project's aim was to test agents in a cooperative Multi-Agent environment, where Agents would have to communicate in order to solve a variety of puzzles. This experiment would be interesting in showing, not only how the agents learn how to solve these new puzzles, but also, exhibit social behaviour in the agents. Another criticism to point out is the amount of episodes done for each comparison, due to some limitations when experimenting. While 150 episodes does show a trend, a larger amount of episodes, 500 to 1000 at least, would properly confirm the aforementioned trends.

7 REFERENCES

https://en.wikipedia.org/wiki/Q-learning
https://www.baeldung.com/cs/epsilon-greedy-q-learning
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
https://pytorch.org/tutorials/intermediate/mario$_rl_tutorial.html$
https://serokell.io/blog/deep-learning-and-neural-network-guidecomponents-of-neural-networks