

UNIVERSIDADE DO MINHO  
MATEMÁTICA E COMPUTAÇÃO

# Reinforcement Learning para jogos simples

Projeto Integrado  
2021/2022

---

*Realizado por:*

Diogo Costa	<b>PG38898</b>
João Dias	<b>PG46799</b>
Luís Araújo	<b>PG46753</b>
Ricardo Teixeira	<b>PG45271</b>

---

25 de junho de 2022

## Resumo

O objetivo deste projeto é estudar e aplicar modelos de *Reinforcement Learning* (**RL**) em jogos simples, utilizando o algoritmo *Q-Learning* em ambientes com um ou mais agentes (jogadores).

O dilema de *Exploration versus Exploitation* é um ponto fulcral em modelos RL, uma vez que, ao não se tratarem de modelos supervisionados, os Agentes (jogadores) precisam de conhecer suficientemente bem o ambiente (jogo) para que eventualmente consigam tomar boas decisões. Parte significativa da nossa implementação passa por estudar e criar técnicas e métricas que ajudem a encontrar o melhor *tradeoff* possível entre ambas as formas de agir.

Numa última fase, abordaremos a ligação entre modelos de RL e a Psicologia, mostrando empiricamente como é possível influenciar o comportamento dos agentes através de uma escolha seletiva das rewards.

A implementação prática foi feita em *Python* e os jogos foram desenvolvidos com recurso à biblioteca *pygame*. Todo o código escrito, quer relativo ao jogo, quer à implementação dos modelos e algoritmo, pode ser acedido através do GitHub.

# Conteúdo

<b>1 Reinforcement Learning</b>	<b>3</b>
1.1 <i>Markov Decision Processes</i> (MDP) . . . . .	4
1.2 <i>V &amp; Q functions</i> . . . . .	5
1.3 Aplicações em jogos simples . . . . .	6
<b>2 Ambiente</b>	<b>7</b>
2.1 Pong . . . . .	7
2.2 Enquadramento com MDP . . . . .	7
<b>3 Agente</b>	<b>10</b>
3.1 Q-matrix & <i>policy</i> . . . . .	10
3.2 <i>Temporal Difference Learning</i> . . . . .	11
3.3 Q-learning . . . . .	11
3.3.1 Exploration vs Exploitation . . . . .	12
<b>4 Métricas de avaliação da <i>Q-matrix</i></b>	<b>15</b>
4.1 <i>Value function</i> . . . . .	15
4.2 <i>Rewards moving average</i> . . . . .	16
4.3 <i>Maximum Q values</i> . . . . .	16
4.4 <i>State visits</i> . . . . .	16
4.5 <i>Best action per state</i> . . . . .	16
<b>5 Treino</b>	<b>19</b>
5.1 $10 \times 10$ . . . . .	19
5.2 $20 \times 20$ . . . . .	20
5.3 $40 \times 40$ e $70 \times 70$ . . . . .	20
<b>6 Múltiplos Agentes</b>	<b>25</b>
6.1 Markov Games . . . . .	25
6.2 Psicologia . . . . .	25
6.3 Dois Agentes . . . . .	26
6.4 Quatro Agentes . . . . .	27
6.5 Discussão de resultados . . . . .	27

# Listas de Figuras

1.1	<i>Framework de Reinforcement Learning</i>	4
2.1	Exemplo de <i>Ambiente</i> do Pong $40 \times 40$	8
4.1	Exemplo de uma <i>counter matrix</i>	17
4.2	Exemplo de <i>best action state</i> para uma <i>grid</i> $10 \times 10$	18
5.1	Exemplo de métricas de avaliação da <i>Q-matrix</i> - <i>grid</i> $10 \times 10$ , método <i>greedy</i>	20
5.2	Exemplo de métricas de avaliação da <i>Q-matrix</i> - <i>grid</i> $20 \times 20$ , método <i>greedy</i>	21
5.3	Exemplo de métricas de avaliação da <i>Q-matrix</i> - <i>grid</i> $20 \times 20$ , método <i>window local greedy</i>	22
5.4	Métricas de avaliação da <i>Q-matrix</i> correspondente ao último treino - <i>grid</i> $40 \times 40$ , método <i>window local greedy</i>	23
5.5	Métricas de avaliação da <i>Q-matrix</i> correspondente ao último treino - <i>grid</i> $70 \times 70$ , método <i>window local greedy</i>	24
6.1	Exemplo de ambiente com 4 agentes	28
6.2	Exemplo de ambiente com 2 agentes	28
6.3	Agente 1, ambiente de 2 agentes	29
6.4	Agente 2, ambiente de 2 agentes	30
6.5	Agente 1, ambiente 4 agentes	31

# Capítulo 1

## Reinforcement Learning

*Reinforcement Learning (RL)* é um dos três paradigmas básicos de *Machine Learning* e é usado para a resolução de problemas onde está associada a escolha de uma ação a realizar.

Neste paradigma existem quatro conceitos chave: **agente**, **ambiente** (expresso como um *Markov Decision Process*), **ação** e **reward**. O objetivo do *RL* é maximizar a *reward* atribuída a uma sequência de ações determinada pelo agente num certo ambiente dinâmico e, para tal, é necessário que o agente "aprenda" como tomar a decisão ótima, seguindo os seguintes passos em instantes de tempo discretos (Fig. 1.1):

- O agente observa o estado do ambiente  $s_t \in S$  no instante de tempo  $t$
- O agente produz uma ação  $a_t \in A(s_t)$  no instante de tempo  $t$
- O ambiente transita para um estado  $s_{t+1} \in S$  e produz uma *reward*  $r_{t+1}$  no instante de tempo  $t + 1$
- O agente observa o estado do ambiente  $s_{t+1}$  e a *reward*  $r_{t+1}$  no instante de tempo  $t + 1$
- ...

Este ramo de *Machine Learning* é muito utilizado em áreas como a robótica, teoria de jogos, teoria de controlo, teoria de informação, entre outros.

Para este projeto, o objetivo é treinar uma inteligência artificial (*AI*) que consiga jogar uma versão ligeiramente diferente e mais simples do jogo tradicional *Pong*. Nesta versão, o jogo será *single player* e o objetivo é manter a bola dentro do domínio, fechado em cima e nos lados por paredes, e aberto em baixo, onde estará um *paddle* controlado pelo *AI* que deverá mover-se de modo a que a bola colida com o mesmo.

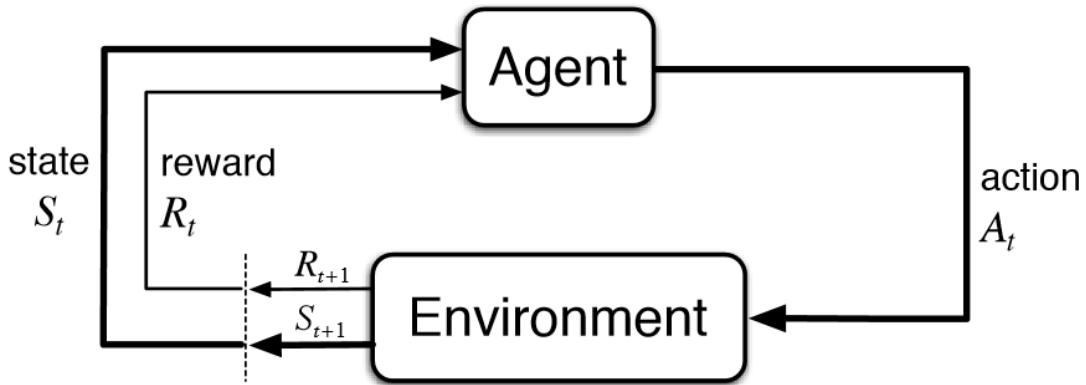


Figura 1.1: *Framework de Reinforcement Learning*

## 1.1 *Markov Decision Processes* (MDP)

Um MDP é um tuplo  $(S, A, P, R, s_0, \gamma)$  onde:

- $S$  é o conjunto de estados (chamado espaço de estados)
- $A$  é o conjunto de ações possíveis (chamado espaço de ações)
- $P_a$  são as probabilidades de transição de um certo estado  $s$  para outro estado  $s'$  através de uma ação  $a$  do espaço de ações  $A$
- $R$  é a lei de probabilidades de *reward* (atribui uma recompensa/penalização dependendo de uma ação  $a$  e uma transição de  $s$  para  $s'$  através de  $a$ )
- $s_0$  é o estado inicial
- $\gamma$  é o *discount rate* da *reward* (menor valor de  $\gamma$  implica que as *rewards* a longo prazo têm menor impacto do que *rewards* a curto prazo e vice-versa)

Uma *policy*  $\pi : S \rightarrow A$  é uma função (potencialmente probabilística) e é o motor do agente pelo que às vezes, por si só, é suficiente para determinar o comportamento do mesmo. Em alguns casos a *policy* pode ser apenas uma função simples ou até uma *lookup table* (caso da nossa implementação) pelo que noutros casos pode envolver computações extensas como um processo de procura. O objetivo do MDP é encontrar uma *policy* óptima  $\pi_*$  tal que para cada estado  $s$ , a ação escolhida pelo agente  $a = \pi_*(s)$  é a melhor possível. Para isso escolhemos  $\pi$  que maximiza a soma cumulativa de *discounted rewards* (*reward* multiplicada por um *discount rate*  $0 \leq \gamma \leq 1$ ), tipicamente o valor esperado da soma de *discounted rewards*.

## 1.2 *V & Q functions*

Um elemento comum à grande generalidade dos modelos de RL é a definição de funções de "valor" (*Value functions*). Estas funções têm a finalidade de determinar a capacidade que um agente tem de interagir com o ambiente, num certo estado. Dado o objetivo das *Value Functions*, é claro que a definição das mesma está dependente da *policy* que o agente segue.

**Definição 1.2.1 ( $V_\pi(s)$  - Value Function)** *Dada uma policy  $\pi$ ,  $\mathbb{E}_\pi$  denota o valor esperado de uma variável aleatória assumindo que um agente segue a policy  $\pi$  e seja  $M = (S, A, P, R, s_0, \gamma)$  uma MDP, para qualquer  $s \in S$ :*

$$V_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s \right]$$

ou equivalentemente (*Bellman Equation*):

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \cdot \sum_{s' \in S} P(s'|s, a) \cdot (R(s, a) + \gamma \cdot V_\pi(s'))$$

Podemos também atribuir uma certa valoração à escolha de uma ação num certo estado:

**Definição 1.2.2 ( $Q_\pi(s, a)$  - Quality Function)** *Dada uma policy  $\pi$ ,  $\mathbb{E}_\pi$  denota o valor esperado de uma variável aleatória assumindo que um agente segue a policy  $\pi$  e seja  $M = (S, A, P, R, s_0, \gamma)$  uma MDP. Dados  $s \in S$  e  $a \in A$ , temos que:*

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \middle| S_t = s, A_t = a \right]$$

ou equivalentemente (*Bellman Equation*):

$$Q_\pi(s) = \sum_{a \in A} P(s|s', a) \cdot \left[ R(s, a, s') + \gamma \cdot \sum_{a' \in A} \pi(a'|s') \cdot Q_\pi(s', a') \right]$$

Estas duas funções estão estreitamente relacionadas, e podemos definir uma a partir da outra:

$$Q_\pi(s, a) = \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma \cdot V_\pi(s')]$$

e  $V_\pi(s)$  a partir de  $Q_\pi$ :

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \cdot Q_\pi(s, a)$$

Dado um agente que segue uma *policy*  $\pi$ , através da experiência (i.e. interação com o ambiente pelo mesmo) sabemos que é possível aproximar o valor das funções  $V_\pi$  e  $Q_\pi$ , uma vez que, para um certo estado, a média das *rewards* acumuladas deverá convergir para o valor de  $V_\pi$  e a média de cada ação  $a$  em cada estado  $s$  deverá convergir similarmente para  $Q_\pi(s, a)$ , à medida que o número de visitas a cada estado tende para infinito.

### 1.3 Aplicações em jogos simples

Uma das técnicas mais investigadas e desenvolvidas de ML atualmente é o *Supervised Learning*, onde cada evento do *Dataset* de treino é previamente etiquetado com a ação/previsão correta para cada um deles. Este tipo de modelos tem como objetivo tentar generalizar e adaptar-se a eventos possíveis mas não presentes nos dados de treino, contudo este tipo de método é muitas vezes impraticável quer por não se saber com exatidão qual a ação/previsão ótima, quer por existir um número extremamente elevado de estados possíveis e muitas vezes com dependência temporal. Este tipo de casos é bastante comum quando queremos treinar um agente para aprender a jogos, mesmo que simples, e se torna necessário aprender via interação. Por outro lado, o *RL* também não é considerada uma técnica de *Unsupervised Learning*, apesar de os dados de treino não estar previamente etiquetados, uma vez que existe uma grande diferença em termos de objetivo. Técnicas de **Unsupervised Learning** tem como objetivo tentar generalizar dados não etiquetados, tentado encontrar estrutura nos mesmos, já o **RL** tem como finalidade maximizar a acumulação de *rewards*.

A caráter dependente de interação do *RL* é o que o torna bastante atrativo para tornar agentes capazes de jogar. No caso em que um Agente apenas tem de interagir com o ambiente (sem adversários) o mesmo apenas está dependente do seu comportamento e das alterações que provoca para maximizar as *rewards*, mas é no caso da existência de oponentes e/ou outros Agentes que o auxiliam (equipas) que emergem as relações mais interessantes entre modelos *RL* e a Psicologia. Neste projeto iremos abordar ambos os casos e avaliar resultados empiricamente.

# Capítulo 2

## Ambiente

Contrariamente a outros modelos de *Machine Learning*, o *Reinforcement Learning* tem como abordagem para alcançar o objetivo predefinido a interação. **Interação**, neste contexto, resume-se ao processo de **observação** de um **Ambiente** e consequente **decisão/ação** por um determinado *Agente* (AI). Na nossa aplicação em concreto, o Agente tem como objetivo aprender a jogar *Pong*, controlando um *paddle* de modo a que a bola se mantenha dentro de uma *grid* o máximo de jogadas possíveis. Em cada estado, o Agente apenas tem informação acerca das dimensões da *grid*, da posição do *paddle* e da posição da bola.

### 2.1 Pong

Devido às características do modelo de *Q-Learning* que estamos a estudar, tivemos a necessidade de criar um *Ambiente* de jogo discreto. Definimos então o *Ambiente* como uma *Grid*  $N \times M$ , a posição de uma única bola na *Grid* dada pelas suas coordenadas  $(x, y) \in \{0, 1, \dots, N - 1\} \times \{0, 1, \dots, M - 1\}$ , e a posição do *paddle* a controlar pelo agente dada por  $(x, M - 2), x \in \{0, 1, \dots, N - pad\_size - 1\}$ . Tanto a bola como o *paddle* têm associados um vetor que indica a sua velocidade, direção e sentido. Durante a execução do jogo, quando a bola colide com o *paddle* um certo número de vezes ( $\lambda$ ) sem que a bola saia do domínio do jogo, consideramos que o agente ganhou o jogo e damos *reset* ao ambiente (retorna ao estado inicial).

### 2.2 Enquadramento com MDP

Fazendo agora referência à definição de  $\text{MDP} = (S, A, P, R, s_0, \gamma)$  como tuplo (secção 1.1) temos:

- $S = \{(x_{bola}, y_{bola}, x_{paddle}) | x_{bola} \in \{0, \dots, N - 1\}, y_{bola} \in \{0, \dots, M - 1\}, x_{paddle} \in \{0, \dots, N - pad\_size - 1\}\}$  que correspondem às posições possíveis da bola  $(x, y)$  e às posições possíveis do *paddle*  $(x, M - 2)$

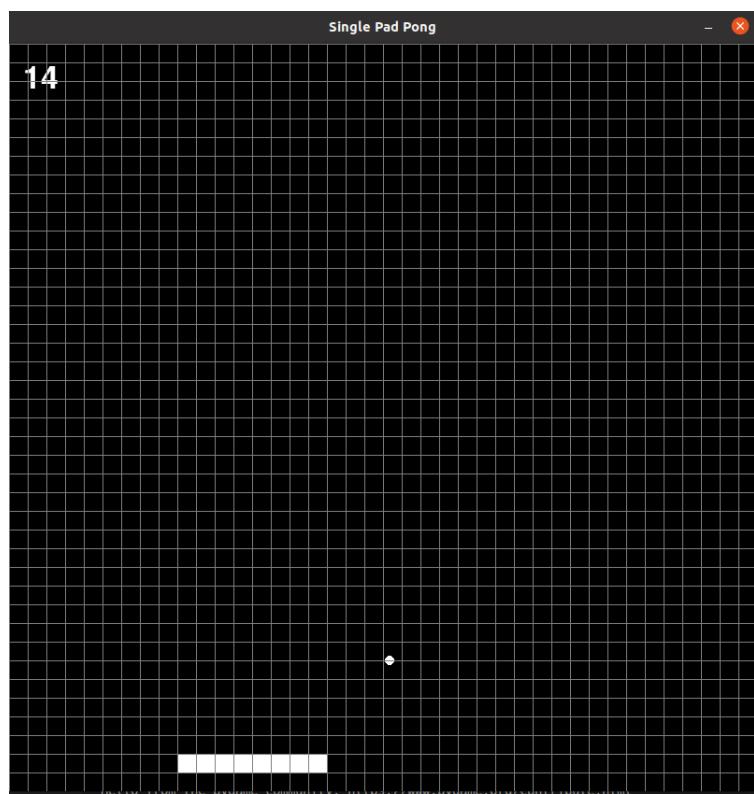


Figura 2.1: Exemplo de *Ambiente* do Pong  $40 \times 40$

- $A = \{não\ mover, mover\ para\ a\ esquerda, mover\ para\ a\ direita\}$  que correspondem aos movimentos possíveis para o *paddle*
- Visto que o jogo é determinístico,  $P_a$  passa a ser uma função de transição que para cada estado  $s$  devolve um e um só estado  $s'$  para qual o ambiente transita através da ação  $a$
- Pelo mesmo motivo que o item anterior,  $R$  é uma função que para cada triplo  $(s, a, s')$  devolve uma *reward*  $r$  positiva caso a transição indicada pelo triplo implique uma colisão da bola com o *paddle*, negativa caso a transição implique a saída da bola do domínio do jogo ou neutra em qualquer outro caso
- $s_0 = (x_{ball}, \lfloor M/2 \rfloor + 1, x_{paddle})$  onde  $x_{ball} \in \{0, \dots, N-1\}$  e  $x_{paddle} \in \{0, \dots, N-1\}$
- O *discount rate* da *reward*  $\gamma$  foi fixado a 0.97 ao longo de todo o projeto.

# Capítulo 3

## Agente

### 3.1 Q-matrix & *policy*

Como vimos anteriormente, é possível estimar os valores da funções  $V_\pi$  e  $Q_\pi$  através da experiência e existem várias técnicas para o realizar. No caso do nosso jogo, caso queiramos estimar o valor de uma destas funções, o mais natural é pensar em atualizações *frame a frame*, ou seja, *one-step updates*. Dentro desta estratégia, é possível definir três subtipos (binários) que se podem combinar:

1. Atualização de **estado** ou de **ação** ( $V$  e  $Q$  respetivamente).
2. Aproximar o valor da **policy ótima** ou para uma dada **policy arbitrária**.
3. Atualizações baseadas em todas as possibilidades de eventos possíveis a partir de um estado ou apenas em *samples* dessas possibilidades.

Dada a natureza discreta e determinista do nosso ambiente de jogo, o subtipo **3** mais adequado ao nosso caso, dentro desta estratégia *one step*, é o de atualizações através de *sampling*. Relativamente ao subtipo **1** e **2** optamos por implementar um modelo baseado em aproximação do valor das ações (1) para uma policy arbitrária (2).

Definida a estratégia, a implementação prática a que recorremos resume-se à construção de uma tabela de *lookup* que representa/aproxima a função  $Q_*$ <sup>1</sup>, à qual chamamos *Q-Matrix*.

---

<sup>1</sup> $Q_*$  denota a função Q seguindo uma *policy* arbitrária e  $Q_\pi$  seguindo a *policy* ótima.

## 3.2 Temporal Difference Learning

Duas técnicas bastante conhecidas para aproximar os valores das funções  $Q$  e  $V$  são *Monte Carlo* (MC) e *Programação Dinâmica* (PD). No caso da PD, é feita uma aproximação de  $V_\pi$  através da *Bellman Equation*, contudo é necessário conhecimento completo do modelo (i.e. das funções  $P_a$  e  $R$  do MDP), algo que não acontece na vasta maioria dos casos. Já MC é uma técnica *Model Free* ou seja é agnóstica ao modelo, e o valor da função é aproximado através de *samples* da função de *reward*  $R$ .

Um terceiro método, aplicado no caso da nossa implementação, é o *Temporal Difference* (TD). TD é *Model Free* e aproxima  $V$  através de *sampling* analogamente ao MC. Contudo, e neste ponto contrariamente a MC, o TD não necessita de observar uma sequência finita de  $n$  *samples* de transições de estados, para calcular a sua média, este método utiliza a última aproximação realizada para um certo estado e atualiza-a (*bootstrap*), formalmente temos [3]:

$$V_k(s_t) := (1 - \alpha_k) \cdot V_{k-1}(s_t) + \alpha_k \cdot (r_t + \gamma V_{k-1}(s_{t+1}))$$

onde  $\alpha_t$  denota o valor do parâmetro auxiliar *StepSize* num dado tempo  $t$ , onde o índice  $k$  representa o valor associado ao  $k$ -ésimo *sample*, mas como no nosso caso a atualização é feita estado a estado,  $k$  e  $t$  coincidem.

## 3.3 Q-learning

A Estratégia utilizada, cujo objetivo é estimar a função  $Q$  para uma *policy* arbitrária ( $Q_*$ ), através de *sample updates*, ganhou na literatura o nome de *Q-Learning* (Watkins 1989).

*Q-Learning* é um algoritmo para estimar  $Q_*$  através de uma *policy* arbitrária, ou seja, independente da *policy* que é seguida o algoritmo garante a convergência desde que todos os estados nunca deixem de ser atualizados.

Como vimos anteriormente, as funções  $Q$  e  $V$  estão profundamente relacionadas, e é dessa ligação que surge o ponto principal do Q-learning, ou seja, transformando a definição da última secção da função  $V$  para TD na função  $Q$ , obtemos:

$$\begin{aligned} V_k(s_t) &:= (1 - \alpha_k) \cdot V_{k-1}(s_t) + \alpha_k \cdot (r_t + \gamma V_{k-1}(s_{t+1})) \\ \implies Q_\pi(s, a) &= R(s, a, s') + \gamma \sum_{s'} P(s, a, s') \sum_{a'} \pi(s', a') Q_\pi(s', a') \end{aligned}$$

e obtemos então a versão de atualização para TD da função  $Q$ , que é a base do *Q-learning* e a fórmula para atualização dos valores da Q-Matrix que representa  $Q$  no caso da nossa implementação:

$$Q_k(s, a) = (1 - \alpha_k) \cdot Q_{k-1}(s, a) + \alpha_k \cdot [R(s, a, s') + \gamma Q_{k-1}(s', a')]$$

E temos então um algoritmo capaz de aprender uma *policy* ótima independente da que é seguida, desde que todas as ações em todos os estados continuem sempre a ser escolhidas (mesmo que com menos frequência):

---

**Algorithm 1** Algoritmo Q-Learning - 1 Agente

---

```

1: for episode = 1, 2, ... do
2:   Observar o estado do ambiente s 2
3:   Escolher e aplicar uma ação a seguindo uma policy  $\pi$  arbitrária. 4
4:   Observar o novo estado do ambiente s' 15
5:   Receber a reward r 17
6:   
$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a')] \quad 19$$

7: end for

```

---

```

1
2 state = ((self.paddle.x//WIDTH_SCALE), (self.ball.y//HEIGHT_SCALE),
3           (self.ball.x//WIDTH_SCALE))
4
5 action = q_ai.action_chooser_method(state, Action_method,
6                                     visits_threshold)
7
8 # right
9 if action == 2:
10    self.game.paddle.move(False, True, window_width=WIDTH)
11 # left
12 elif action == 1:
13    self.game.paddle.move(False, False, window_width=WIDTH)
14
15 game_info = self.game.loop()
16
17 new_state = ((self.paddle.x//WIDTH_SCALE), (self.ball.y//HEIGHT_SCALE),
18               (self.ball.x//WIDTH_SCALE))
19 r = self.reward(init_score, end_score)
20
21 q_ai.q(action, r, state, new_state,
22         negative_propagation=negative_propagation)

```

### 3.3.1 Exploration vs Exploitation

Num cenário em que um algoritmo pudesse correr por tempo indeterminado, a teoria garante-nos a convergência, contudo, o nosso objetivo é tentar fazer com que a Matriz de um Agente converga o mais rápido possível para  $Q_*$ . Posto isto, enfrentamos o mais recorrente e complexo *tradeoff* entre modelos de RL, o dilema

entre *exploration* (i.e. o quanto um agente deve explorar o ambiente - testar ações de forma total ou parcialmente agnóstica à performance) e *exploitation* (i.e. se o agente deve apenas escolher as ações com melhores valores na  $Q$  – Matrix).

Para abordar este problema, utilizamos duas estratégias de exploração, e duas *policies* para cada uma delas:

1. Estratégias de exploração foco local:

- $\pi_{wlg,r}$  Window local Greedy
- $\pi_{leg}$  Local Epsilon Greedy

2. Estratégias de exploração de foco global:

- $\pi_g$  Greedy
- $\pi_{eg}$  Decaying Epsilon Greedy

**Definição 3.3.1 (Estratégia de exploração local)** *Uma estratégia de exploração local é uma policy  $\pi$  que, para um certo estado  $s$ , retorna uma ação  $a$ , baseando-se no quanto bem  $s$  e/ou estados adjacentes na matriz (associados a zonas adjacentes do ambiente), ou seja, quantas vezes  $s$  ou uma região que envolve  $s$  foi visitada. Para qualquer estado  $s$ , à medida que o número de visitas ao mesmo aumenta, a policy vai diminuindo o seu caráter exploratório e aumentando a probabilidade que a ação escolhida seja a com o maior absoluto em  $Q$  – Matrix( $s$ ).*

**Definição 3.3.2 (Estratégia de exploração global)** *Uma estratégia de exploração global é uma policy  $\pi$  que, para qualquer estado  $s$ , retorna uma ação com base no valor de uma função monótona e/ou decrescente  $f$  cujos parâmetros são o tempo  $t$  e o estado  $s$ . Esta função deverá decrescer (ou potencialmente estagnar) consoante o ambiente vai sendo visitado, ou seja, está intimamente relacionada com:*

$$\sum_{s \in S} Q_c(t, s)$$

*onde  $Q_c(t, s)$  denota a quantidade de vezes que  $s$  foi visitado até ao instante  $t$ , pelo agente.*

**Definição 3.3.3 ( $\pi_g$  Greedy)** *A policy greedy tem o exatamente o mesmo comportamento que a definição clássica, mas dada a definição 3.3.2, pode também ser vista como uma estratégia de exploração global em que a função decrescente é  $f(t, s) = 0$ , ou seja, é sempre escolhida a melhor ação.*

**Definição 3.3.4 ( $\pi_{eg}$  Decaying Epsilon Greedy)** *esta definição é uma caso particular do Decaying Epsilon Greedy tradicional, em que a função descrente associada é o rácio entre*

**Definição 3.3.5 ( $\pi_{wlg}$  Window local Greedy)**  $\pi_{wlg}$  é uma policy que verifica numa janela de adjacência com um certo raio, se, para um certo estado  $s$  os seus vizinhos estão ou não, bem visitados, e explore conforme a média de visitar dessas janelas.

# Capítulo 4

## Métricas de avaliação da *Q-matrix*

Como referido anteriormente, o comportamento do agente é completamente determinado pela qualidade da sua *Q-matrix*. Mas como é que distinguimos uma *Q-matrix* "boa" de uma *Q-matrix* "má"? Para responder a esta pergunta, foi necessário criar várias métricas que vamos definir nesta secção que permitem avaliar a qualidade de uma *Q-matrix* e o quanto próxima esta está de uma *Q-matrix* óptima. Da teoria, sabemos que se todos os estados forem visitados um número infinito de vezes, a *Q-matrix* converge necessariamente para uma matriz que representa  $Q_*$  mas, visto que estamos limitados a uma execução finita, o nosso objetivo passa por encontrar uma *Q-matrix* que se aproxime de  $Q_*$  o mais rápido possível e queremos também quantificar uma *Q-matrix* que permita ao agente consiga jogar, não de uma maneira perfeita, mas da maneira mais satisfatória possível.

### 4.1 *Value function*

A primeira métrica implementada e provavelmente o melhor indicador de convergência para  $Q_*$  é a média da *value-state function*  $V(s)$  (secção 3.1):

$$\frac{1}{|S|} \sum_{s \in S} V_\pi(s)$$

No nosso caso, a política  $\pi$  seguida para um certo estado  $s$  corresponde a escolher a ação cujo valor da *Q-matrix* para esse estado é maior ( $\underset{a}{\operatorname{argmax}} Q(s, a)$ ).

Teoricamente, o valor máximo de  $V_\pi(s)$  é dado por  $\frac{r_{max}}{1-\gamma}$ , ou seja, para *rewards*  $r \in \{-1, 0, 1\}$  e  $\gamma = 0.97$  temos que  $V_{\pi max}(s) = 33.(3)$  e portanto a matriz  $Q_*$  tem, para cada estado, uma ação ótima  $a_*$  tal que  $Q_*(s, a_*) = V_{\pi max}(s)$  e para as outras ações  $a, a'$ ,  $Q(s, a) \approx Q(s, a') \leq 0$

## 4.2 *Rewards moving average*

Seja  $n$  um certo número de episódios (um episódio corresponde a uma transição de estados). Definimos *rewards moving average* da seguinte forma:

$$rma = \frac{1}{n} \sum_{i=1}^n r_i$$

Esta métrica pertite-nos avaliar a aptidão do agente de jogar. Para uma matriz que representa  $Q_*$ , o valor da  $rma$  é  $r_{max}$  mas o inverso não é equivalente, ou seja, se  $rma = r_{max}$  apenas podemos concluir que o comportamento do agente é bom o suficiente para jogar sem perder, mesmo que a matriz não se aproxime muito da matriz de  $Q_*$ .

## 4.3 *Maximum Q values*

Esta métrica foi implementada devido a uma dificuldade que encontramos em ambientes com pouca complexidade (por exemplo dimensão da grelha pequena), pois verificamos que, por vezes, poucos valores ou nenhum convergiam para o valor ótima esperado. Apenas indica o valor máximo da *Q-matrix* e serve para saber se algum estado já atingiu ou está próximo de atingir o valor ótimo, à medida que o treino evolui.

## 4.4 *State visits*

Seja  $vt$  um *threshold* de visitas a um certo estado e  $S$  o conjunto de estados que foram visitados mais que  $vt$  vezes. A métrica *State visits* é o rácio entre  $|S|$  e  $|\bar{S}|$  e indica a quantidade de estados que ainda não foram suficientemente visitados. Para a implementação desta métrica é usada uma matriz auxiliar que para cada estado, contem o número de visitas (*counter matrix* como por exemplo 4.1).

## 4.5 *Best action per state*

Esta métrica permite-nos identificar qual a melhor ação para cada estado e permite-nos visualizar de uma maneira mais geral qual o comportamento do agente.  $\{R, G, B\} = \{\text{não mover}, \text{mover para a esquerda}, \text{mover para a direita}\}$  ( 4.2)

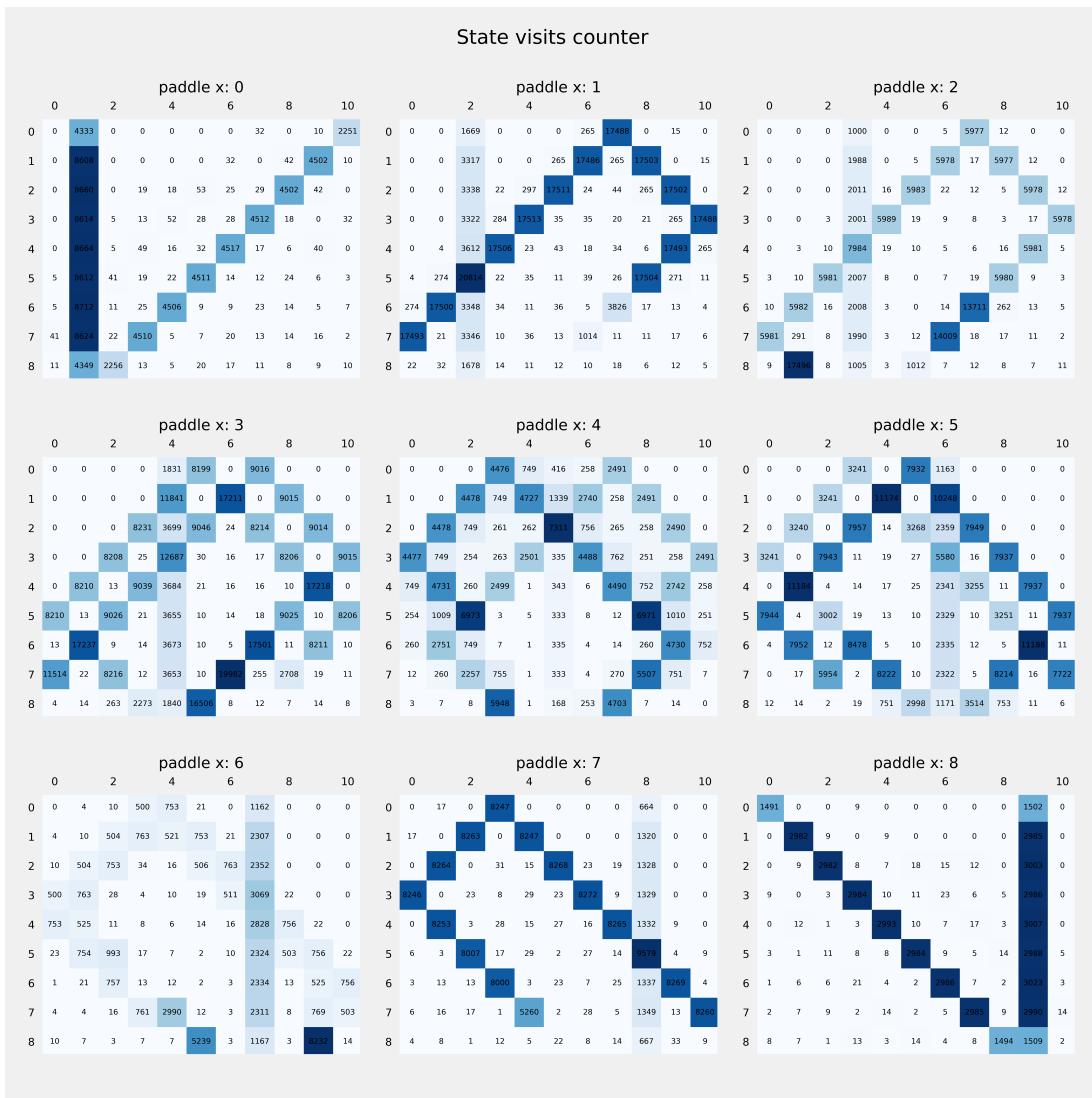


Figura 4.1: Exemplo de uma *counter matrix*

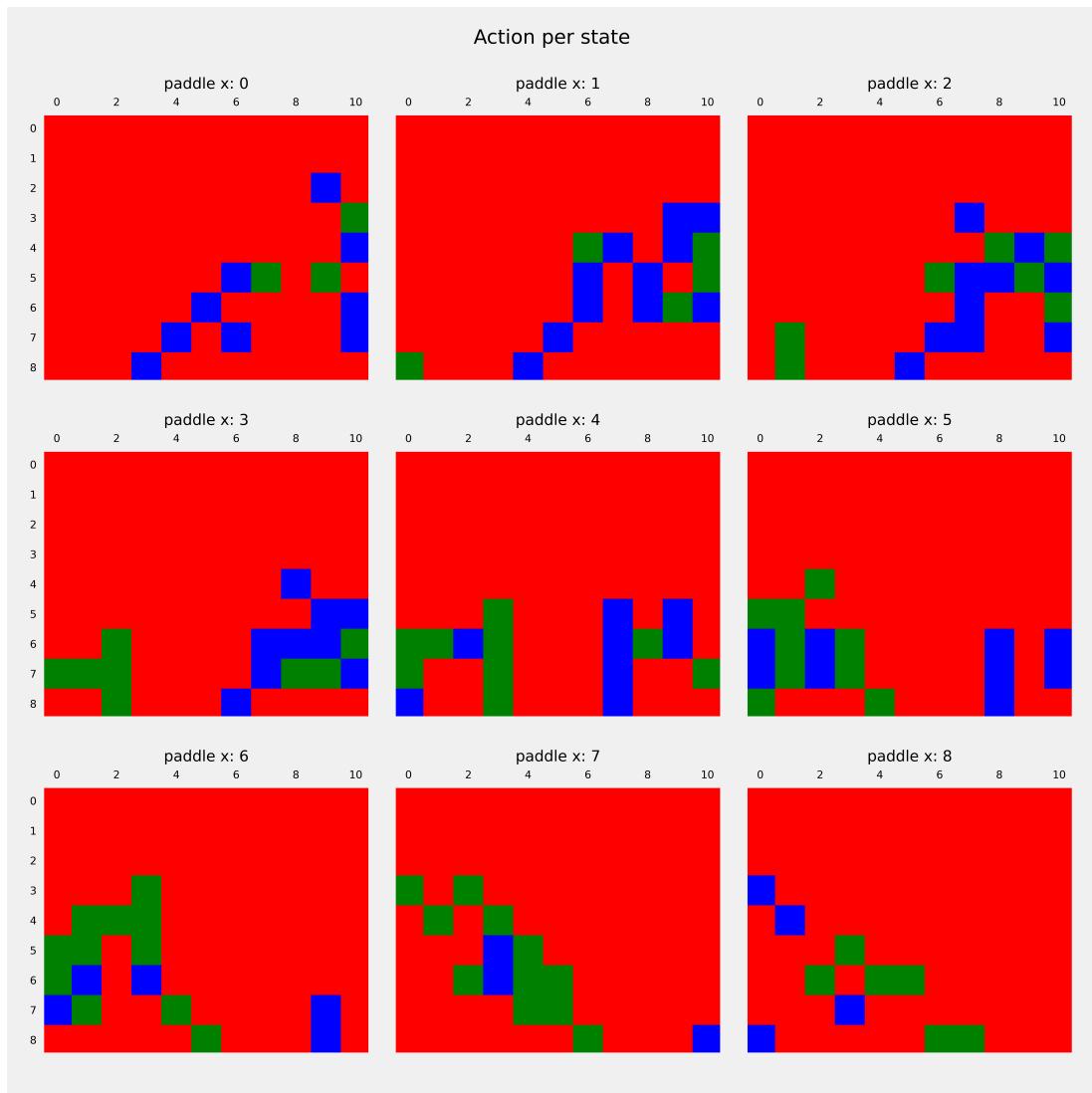


Figura 4.2: Exemplo de *best action state* para uma *grid*  $10 \times 10$

# Capítulo 5

## Treino

O treino consiste em executar um número finito de iterações de um algoritmo de aprendizagem (neste caso o *Q-learning*), estando concluído quando alcançamos um objetivo predefinido. Para este problema havia dois objetivos:

1. Alcançar a matriz ótima que representa a função  $Q_*$
2. Alcançar uma matriz que permita ao agente jogar perdendo um número muito reduzido de vezes

Visto que o primeiro objetivo requer um tempo de treino impraticável e o segundo é algo relativamente simples de realizar, optamos por tentar alcançar um equilíbrio entre ambos os objetivos, ou seja, alcançar uma matriz que permita ao agente perder pouco tendo também em atenção o nível de aproximação da mesma à matriz ótima (matriz da função  $Q_*$ ).

Para isso, à semelhança de outros modelos de *Machine Learning*, foi necessário "afinar" vários parâmetros como o *learning rate*  $\alpha$ , o número de vezes que a bola colide com o *paddle* seguidas até dar *reset* ao ambiente  $\lambda$  e o rácio de *exploitation-exploration* (tanto o método a usar, como os parâmetros de cada método como por exemplo, o método  $\epsilon - greedy$  e o seu parâmetro  $\epsilon$ ). Estas afinações foram feitas usando o método de *random search* (basicamente, *brute force* de combinações de parâmetros) assim como alguma intuição.

Ao fim de cada treino, avaliamos a qualidade do mesmo através das métricas referidas no capítulo anterior, assim como observação do agente a jogar.

### 5.1 $10 \times 10$

Como podemos ver na figura 5.1, conseguimos chegar ao primeiro objetivo de o agente jogar o jogo sem perder usando qualquer um dos métodos de exploração (mesmo para o método *greedy* para o qual a exploração é mínima) visto que o jogo com estas dimensões é extremamente simples e o agente encontra os padrões

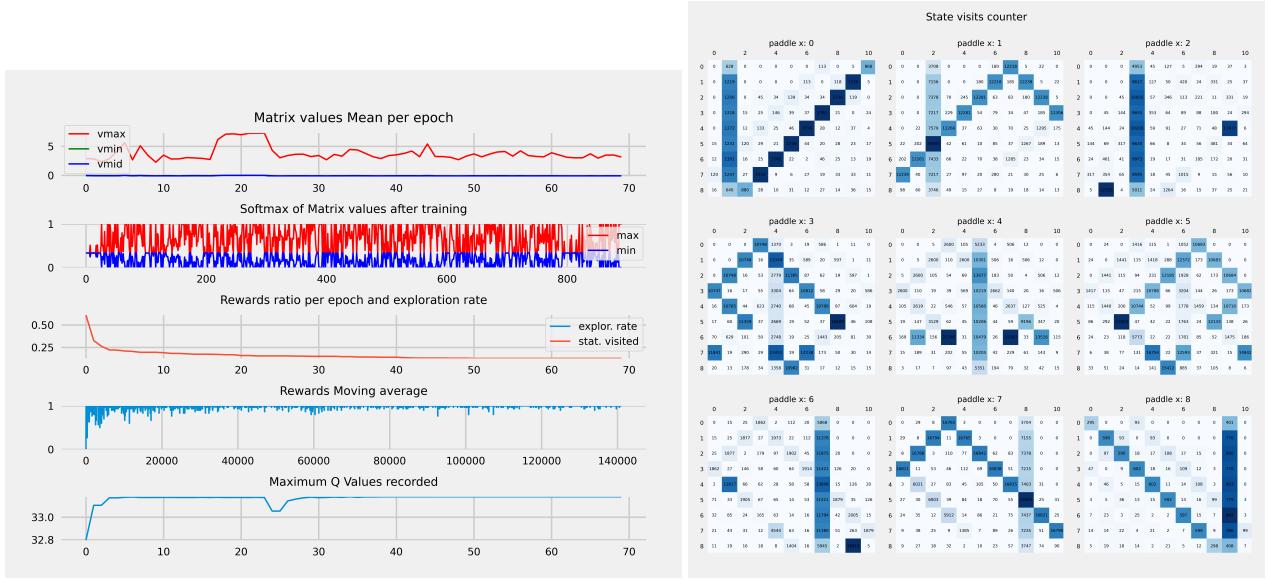


Figura 5.1: Exemplo de métricas de avaliação da *Q-matrix* - grid  $10 \times 10$ , método *greedy*

de movimentos em poucas épocas. Também é relativamente simples de atingir o valor ótimo de  $V_*(s) = 33.(3)$  para alguns estados muito visitados. No entanto verificamos que alguns estados não eram visitados, independentemente de usarmos um nível de exploração muito alto ou muito baixo, pelo que pusemos a hipótese de que esses seriam inatingíveis devido ao movimento da bola. Como de esperar, e também com alguma influência de haver estados possivelmente inatingíveis a impactar as métricas pela negativa, não foi possível de chegar a valores médios de  $Q$  ótimos em tempo útil.

## 5.2 $20 \times 20$

Como podemos observar nos gráficos 5.2 e 5.3, conseguimos na mesma chegar ao ponto em que o agente consegue não perder, no entanto já começamos a observar alguns problemas na escalabilidade do problema. Devido ao aumento da dimensão os valores médios da *Q-matrix* ficam agora ainda mais afastados dos valores ideais e já não conseguimos atingir o valor ótimo de  $V_*(s) = 33.(3)$  apesar do aumento do tempo de treino.

## 5.3 $40 \times 40$ e $70 \times 70$

Para estas dimensões mesmo apesar de efetuarmos múltiplos treinos seguidos, o primeiro objetivo de o agente não perder já foi difícil de atingir como podemos

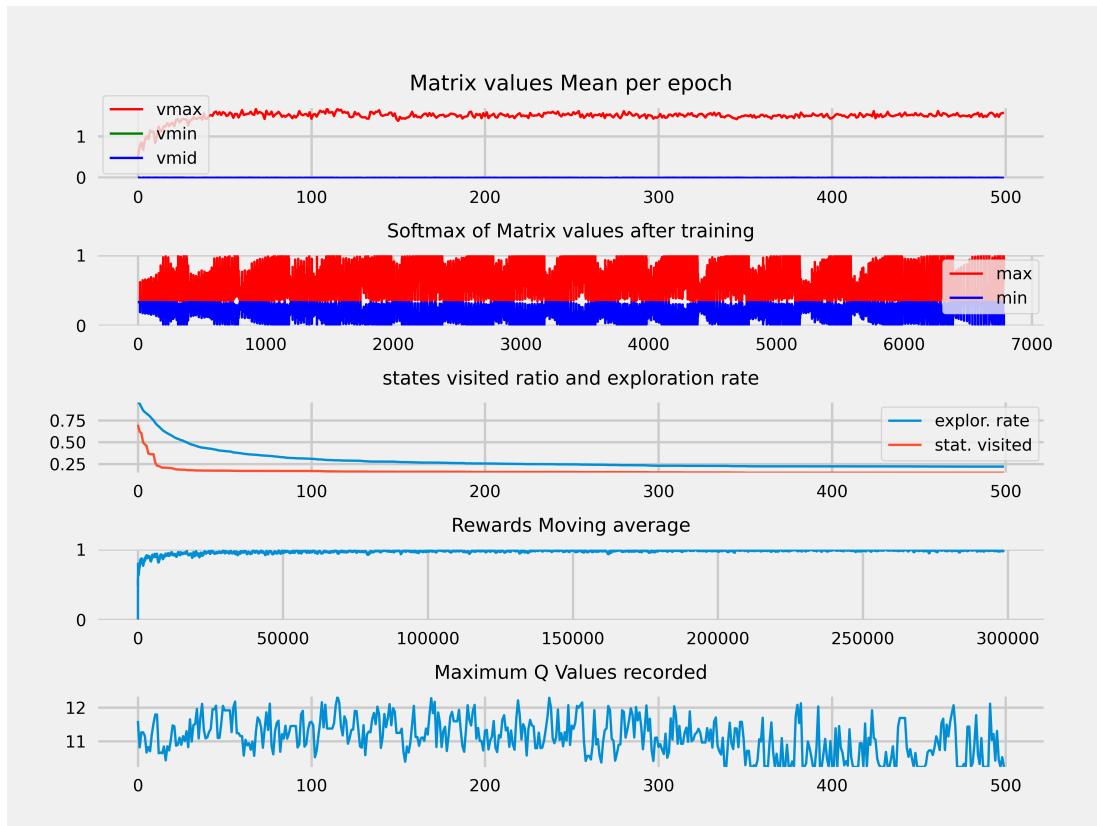


Figura 5.2: Exemplo de métricas de avaliação da  $Q$ -matrix - grid  $20 \times 20$ , método *greedy*

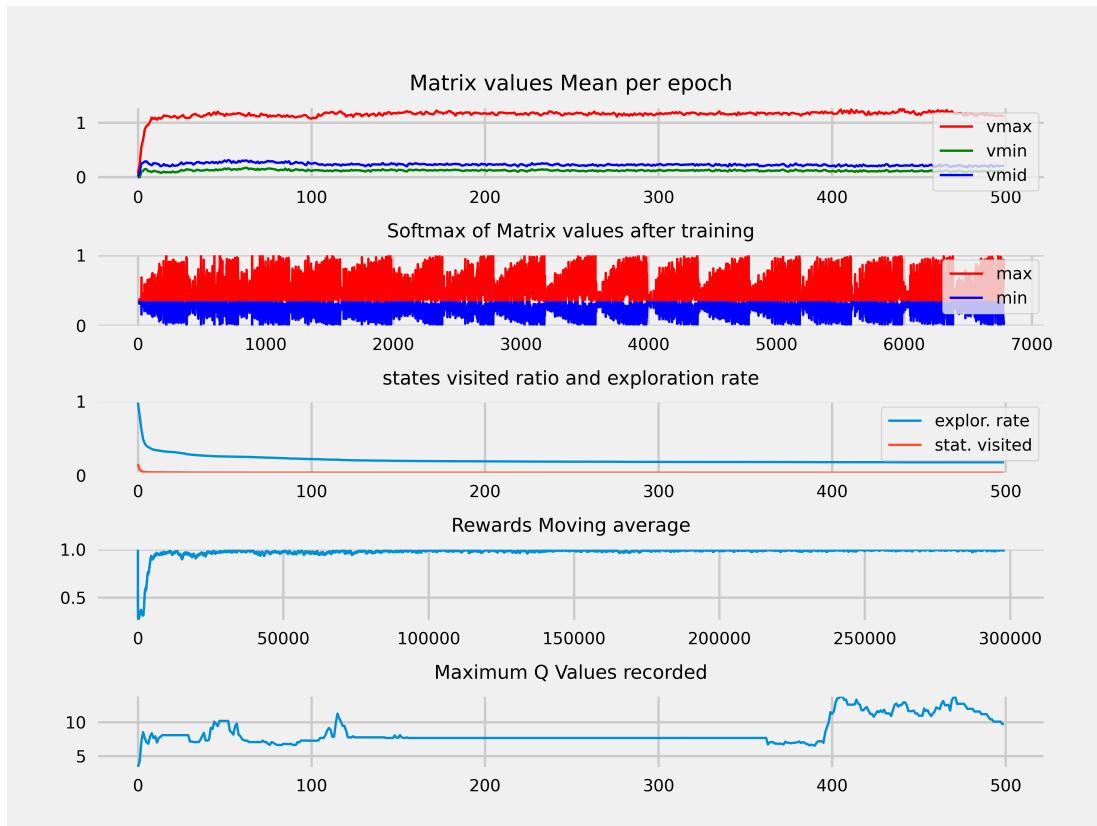


Figura 5.3: Exemplo de métricas de avaliação da  $Q$ -matrix - grid  $20 \times 20$ , método *window local greedy*

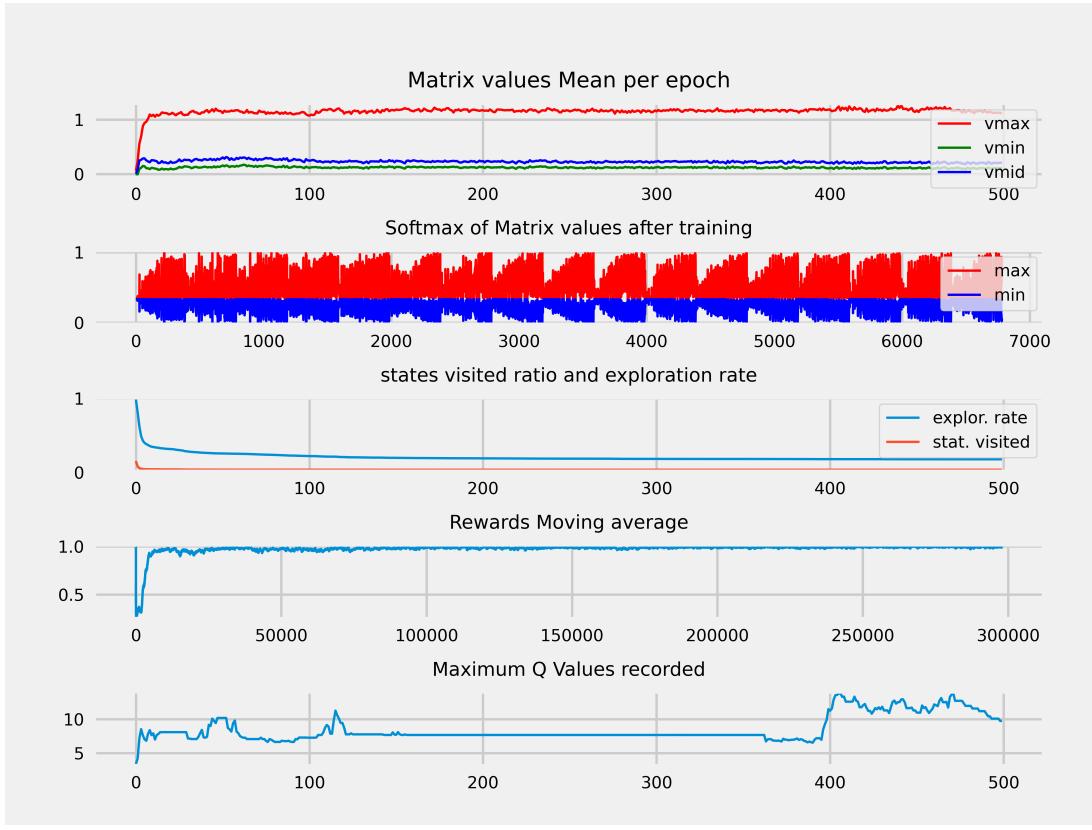


Figura 5.4: Métricas de avaliação da  $Q$ -matrix correspondente ao último treino - grid  $40 \times 40$ , método *window local greedy*

ver nas figuras 5.4 e 5.5. Já quanto à qualidade da  $Q$ -matrix, obtivemos valores médios muito baixos, e mesmo os mais altos eram apenas cerca de  $\frac{1}{3}$  do valor ótimo, pelo que concluímos que já estariámos a chegar aos limites do  $Q$ -learning

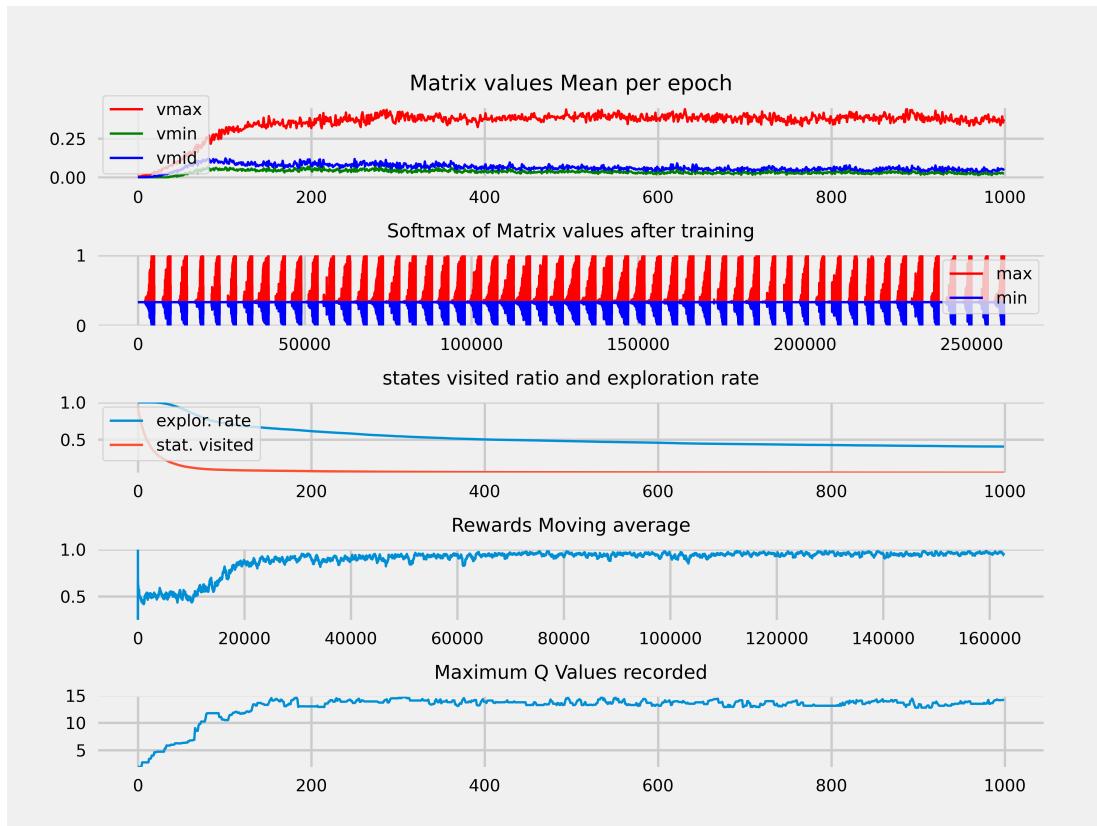


Figura 5.5: Métricas de avaliação da  $Q$ -matrix correspondente ao último treino - grid  $70 \times 70$ , método *window local greedy*

# Capítulo 6

## Múltiplos Agentes

### 6.1 Markov Games

Um MDP é capaz de modelar matematicamente a interação entre um ambiente e um agente, contudo, é interessante o caso onde, num mesmo ambiente, temos múltiplos agentes a interagir. Um *Markov/Stochastic game* MG foi proposto como uma framework standard para modelar a interação entre múltiplos agentes:

**Definição 6.1.1 (*Markov game* MG)** *Um MG é um tuplo  $(S, \mathbb{A}, \mathbb{R}, P, \gamma)$  onde [2]:*

1. *S é o conjunto de estados do ambiente*
2.  *$\mathbb{A} = A_1 \times \dots \times A_n$  é o produto cartesiano dos Conjuntos de ações disponíveis para um agente  $i, i \in \{1, \dots, N\}$ .*
3.  *$\mathbb{R} : S \times \mathbb{A} \times S \longrightarrow \mathbb{R}^N$  é uma função que atribui uma reward a cada um dos agentes.*
4.  *$P : S \times \mathbb{A} \longrightarrow S$  é a função de transição após a execução de todas as ações por todos os agentes.*
5.  *$\gamma$  é o discount factor.*

*De notar que esta formulação está enquadrada com o caso determinístico e discreto do nosso jogo.*

### 6.2 Psicologia

Os Modelos de RL tiveram desenvolvimentos e avanços consideráveis através de ideias já conhecidas da área da Psicologia animal. Contudo, do lado de quem implementa estes modelos, o objetivo normalmente resume-se à construção de

um modelo robusto para resolver um certo problema e nunca o de interpretar o comportamento de um agente de forma agnóstica à sua performance. Neste capítulo visitaremos esta relação interessante entre o RL e a Psicologia, tentando induzir e interpretar certos comportamentos em jogos com vários agentes, através de uma escolha criteriosa de *rewards*.

### 6.3 Dois Agentes

No caso de um ambiente com 2 agentes, é interessante observar dois tipos de comportamentos, um cooperativo (utilitarista [1]) e outro competitivo (ganancioso). Para induzir estes comportamentos utilizamos vários tipos de *rewards*. Para algum estado  $s$  e dado o conjunto de ações tomadas pelos agentes  $\delta \in \Delta$ , definimos as seguintes funções de *reward* para os agentes  $\Delta_1$  e  $\Delta_2$ :

$$R_1(s, a, s') = \begin{cases} (-1, 1) & \text{se } \Delta_1 \text{ deixar passar a bola} \\ (1, -1) & \text{se } \Delta_2 \text{ deixar passar a bola} \\ (0, 0) & \text{c.c.} \end{cases} \quad (6.1)$$

No caso de 6.1, temos um *Zero-sum game* em que se espera que os agentes competam e apenas ganham *reward* quando o outro deixa a bola passar.

$$R_2(s, a, s') = \begin{cases} (-1, -1) & \text{se } \Delta_1 \text{ deixar passar a bola} \\ (-1, -1) & \text{se } \Delta_2 \text{ deixar passar a bola} \\ (0, 0) & \text{c.c.} \end{cases} \quad (6.2)$$

$$R_3(s, a, s') = \begin{cases} (1, 1) & \text{se } \Delta_1 \text{ apanha a bola} \\ (1, 1) & \text{se } \Delta_2 \text{ apanha a bola} \\ (0, 0) & \text{c.c.} \end{cases} \quad (6.3)$$

Em 6.2 e 6.3 espera-se uma cooperação entre os agentes, pois ambos são penalizados quando a bola sai em 6.3, e ambos beneficiados quando um apanha a bola em 6.3.

$$R_4(s, a, s') = \begin{cases} (1, 0) & \text{se } \Delta_2 \text{ deixa passar a bola} \\ (0, 1) & \text{se } \Delta_2 \text{ apanha a bola} \\ (0, 0) & \text{c.c.} \end{cases} \quad (6.4)$$

Por fim temos uma implementação 6.4 que tenta impingir personalidades diferentes a ambos os agentes, de forma a que  $\Delta_1$  seja mais competitivo e tente ganhar sempre e  $\Delta_2$  seja mais defensivo e conservador.

## 6.4 Quatro Agentes

Num jogo com 4 agentes ( $\Delta_i, i \in \{1, 2, 3, 4\}$ ), em que não existem paredes a limitar o ambiente, é interessante modelar as funções de *reward* de forma a tentar formar equipas entre os agentes, temos:

$$R_1^4(s, a, s') = \begin{cases} (0, 0, 1, 1) & \text{se } \Delta_1 \text{ ou } \Delta_2 \text{ deixar passar a bola} \\ (1, 1, 0, 0) & \text{se } \Delta_3 \text{ ou } \Delta_4 \text{ deixar passar a bola} \\ (0, 0, 0, 0) & \text{c.c.} \end{cases} \quad (6.5)$$

A equipa 1 é formada pelos Agentes  $\Delta_1$  e  $\Delta_2$  e a equipa 2 pelos agentes  $\Delta_3$  e  $\Delta_4$ . Em 6.5 tentamos fazer com que ambas as equipas tentem fazer exclusivamente que a outra perca.

$$R_1^4(s, a, s') = \begin{cases} (1, 1, 1, 1) & \text{se } \Delta_1, \Delta_2, \Delta_3 \text{ ou } \Delta_4 \text{ apanhar a bola} \\ (-1, -1, -1, -1) & \text{se } \Delta_1, \Delta_2, \Delta_3 \text{ ou } \Delta_4 \text{ deixar passar a bola} \\ (0, 0, 0, 0) & \text{c.c.} \end{cases} \quad (6.6)$$

Através de 6.6 tentamos impingir uma colaboração de equipa entre os 4 jogadores.

## 6.5 Discussão de resultados

A análise que fizemos para os ambientes de apenas um agente necessita de outra sensibilidade caso a queiramos utilizar nos ambientes multi-agente, uma vez que as dinâmicas do sistema se tornam bastante mais complexas. No caso de dois agentes é interessante verificar que num ambiente competitivo existe sempre um Agente que se consegue sobressair primeiro e, ao fazê-lo, consegue prejudicar a própria aprendizagem do outro agente, como é possível verificar em 6.3 e 6.4.

Já no caso de 4 agentes, é possível criar situações de satisfatórias (de cooperação por exemplo) em que um ou mais agentes têm matrizes com valores muito abaixo do desejável 6.5.

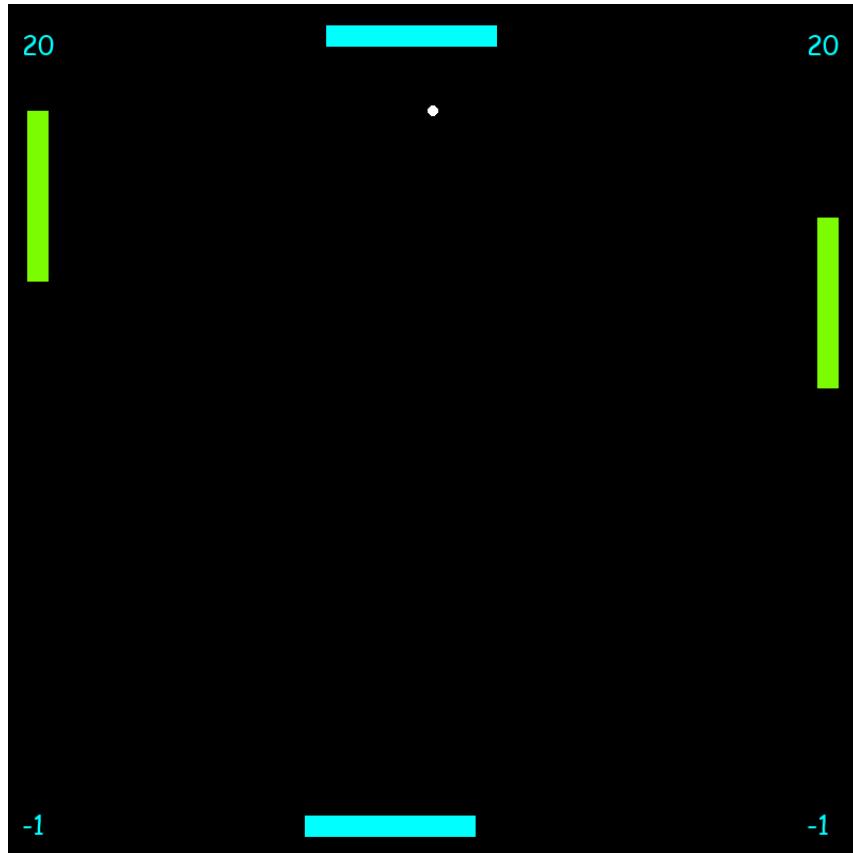


Figura 6.1: Exemplo de ambiente com 4 agentes

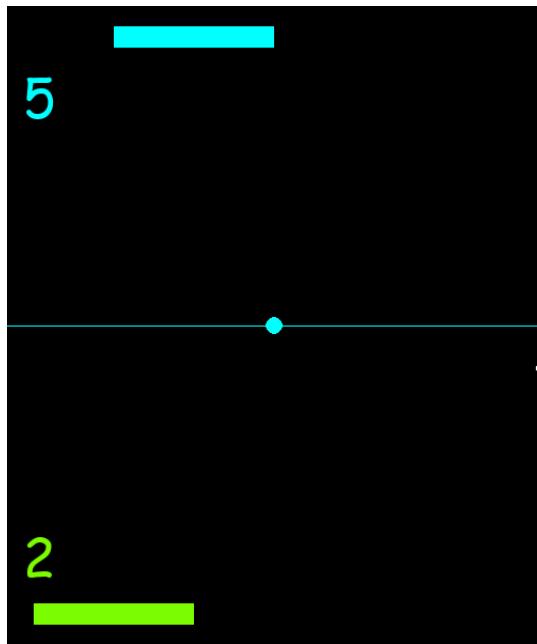


Figura 6.2: Exemplo de ambiente com 2 agentes

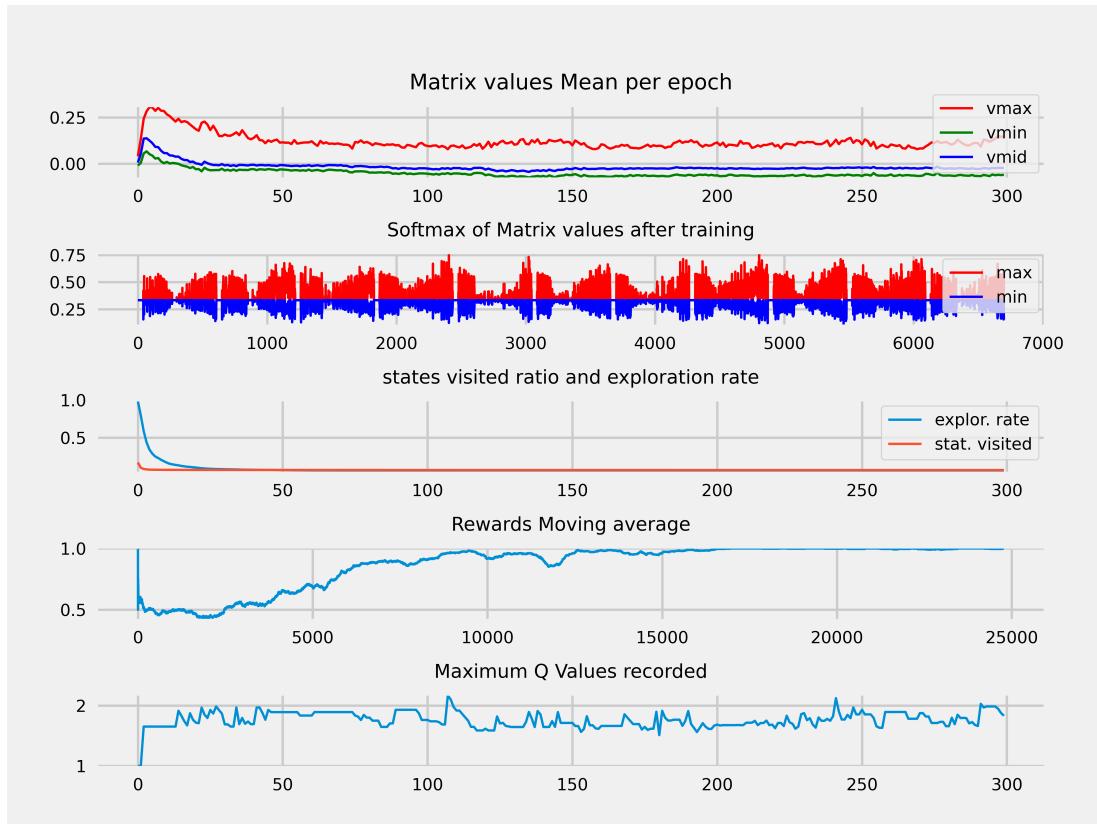


Figura 6.3: Agente 1, ambiente de 2 agentes

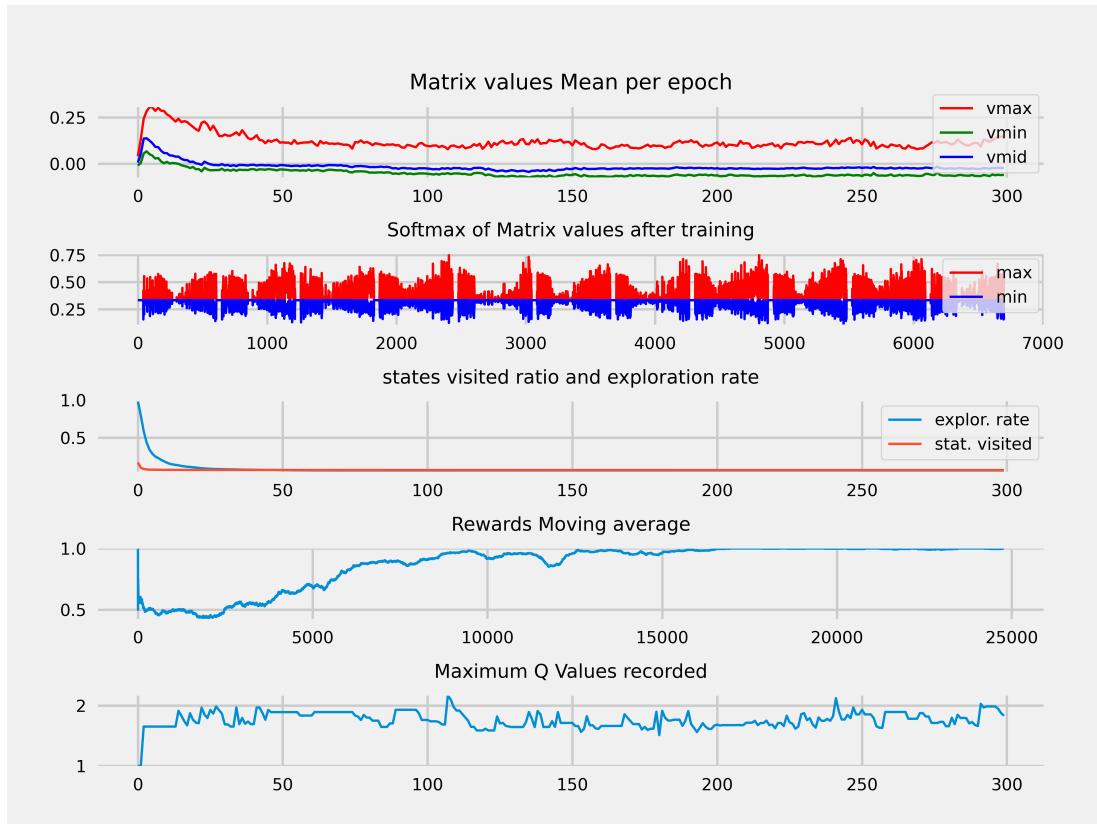


Figura 6.4: Agente 2, ambiente de 2 agentes

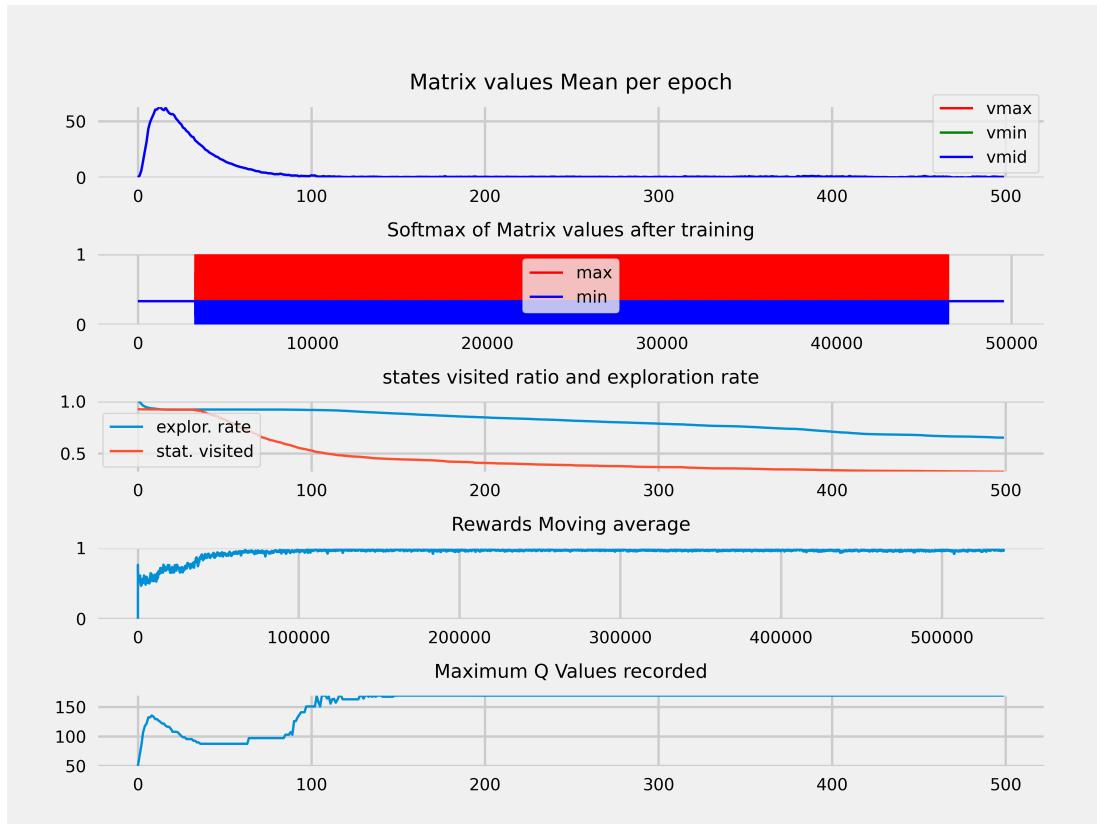


Figura 6.5: Agente 1, ambiente 4 agentes

# Papers

- [1] Pieter Hoen et al. “An Overview of Cooperative and Competitive Multiagent Learning.” Em: (jan. de 2005), pp. 1–46.
- [2] Cyrus Neary et al. “Reward Machines for Cooperative Multi-Agent Reinforcement Learning”. Em: *CoRR* abs/2007.01962 (2020). arXiv: 2007 . 01962. URL: <https://arxiv.org/abs/2007.01962>.

# Livros e Slides

- [3] Richard S Sutton e Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.