UAlg
UNIVERSIDADE DO ALGARVE

Ano letivo 2016/2017

Relatório

Jogo Drop 4

3º Trabalho Pratico

Grupo 05:

Alexandra Brito nº50075

Cyrill Brito nº52875

Índice

Introdução

Este relatório tem como objetivo principal esclarecer e informar todos os métodos e técnicas utilizadas no desenvolvimento do 3º trabalho pratico da disciplina de inteligência artificial, *''Drop 4''*.

Este ultimo trabalho da disciplina era de escolha livre, nos poderíamos escolher qualquer problema que quiséssemos. Na altura da escolha, nos estávamos ambos bastante fascinados com um jogo que tínhamos encontrado para o nossos smartphones, o *''Drop 7'' (1)*. Com esta grande obsessão decidimos utilizar o conceito desde jogo para o este ultimo trabalho da disciplina.

O objetivo do nosso trabalho foi criar um bot que jogasse ao jogo de forma eficaz, aprendendo como se joga por tentativas. Para isso foi utilizado o algoritmo Q-learning que é uma aplicação de *reinforcement learning.* Escolhemos este método porque é um dos melhores algoritmos para a resolução de jogos e não só. E também porque é uma tecnologia já bastante utilizada, como por exemplo a google utilizou a tecnologia para a DeepMind *(3)*.

Inicialmente tínhamos como objetivo criar um clone do jogo Drop 7, este jogo implementa um tabuleiro de 7 por 7, mas devido á complexidade do jogo decidimos diminuir para 4 por 4.

Drop 4

O jogo consiste num tabuleiro de 4x4 e em peças que contêm ou um numero de 1 a 4 ou são em branco. Perde-se o jogo quando é jogada uma peça que já não caiba no tabuleiro 4x4. O jogador vai colocando peças selecionando a coluna, assim como no jogo 4 em linha, as peças também são afetadas por gravidade, ou seja, a peça introduzida cai para a posição livre mais abaixo possível. O objetivo do jogo é rebentar o maior número de peças possíveis antes de perder.

Como referido anteriormente as peças podem ter vários valores, ou ser brancas. As peças com um numero rebentam quando a peça pertence a uma linha, vertical ou horizontal, que tenha o mesmo numero de peças que o numero que está nessa peça.

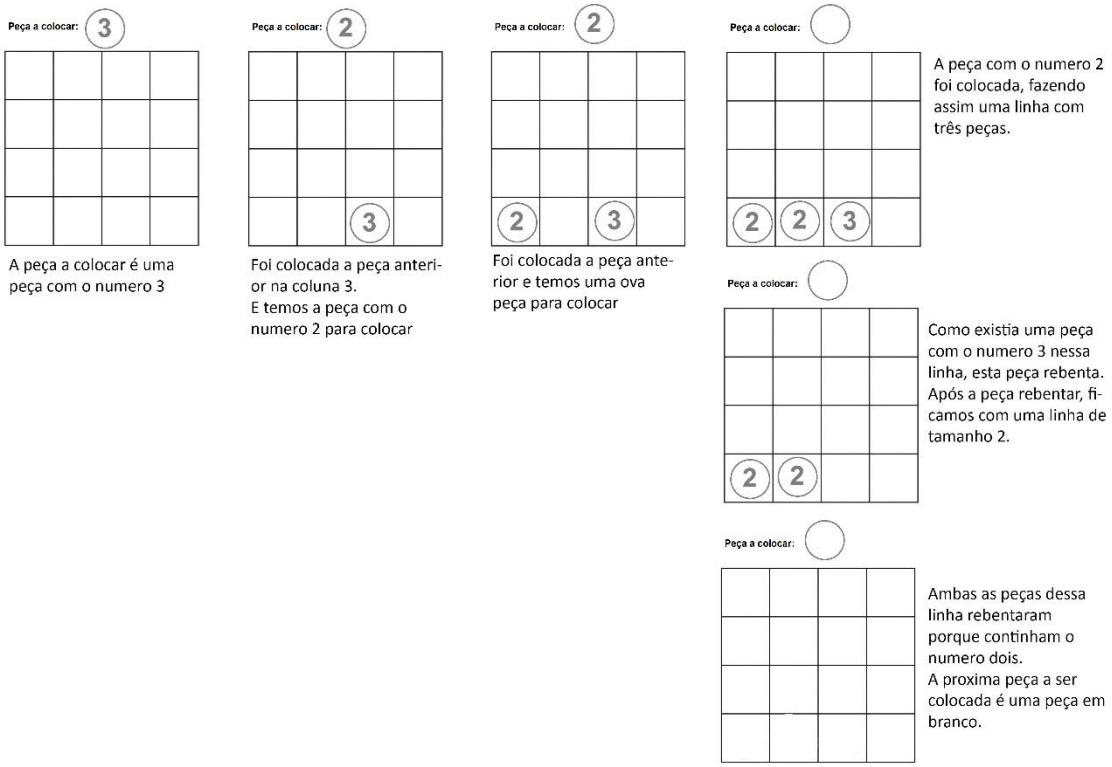

Figura 1 Exemplo do jogo 1

As peças em branco não rebentam, apenas podem se transformar noutra peça, esta nova peça pode ser um numero de 1 a 4 ou ser outra peça em branco. As peças em branco transformam-se quando uma peça rebenta próximo delas.

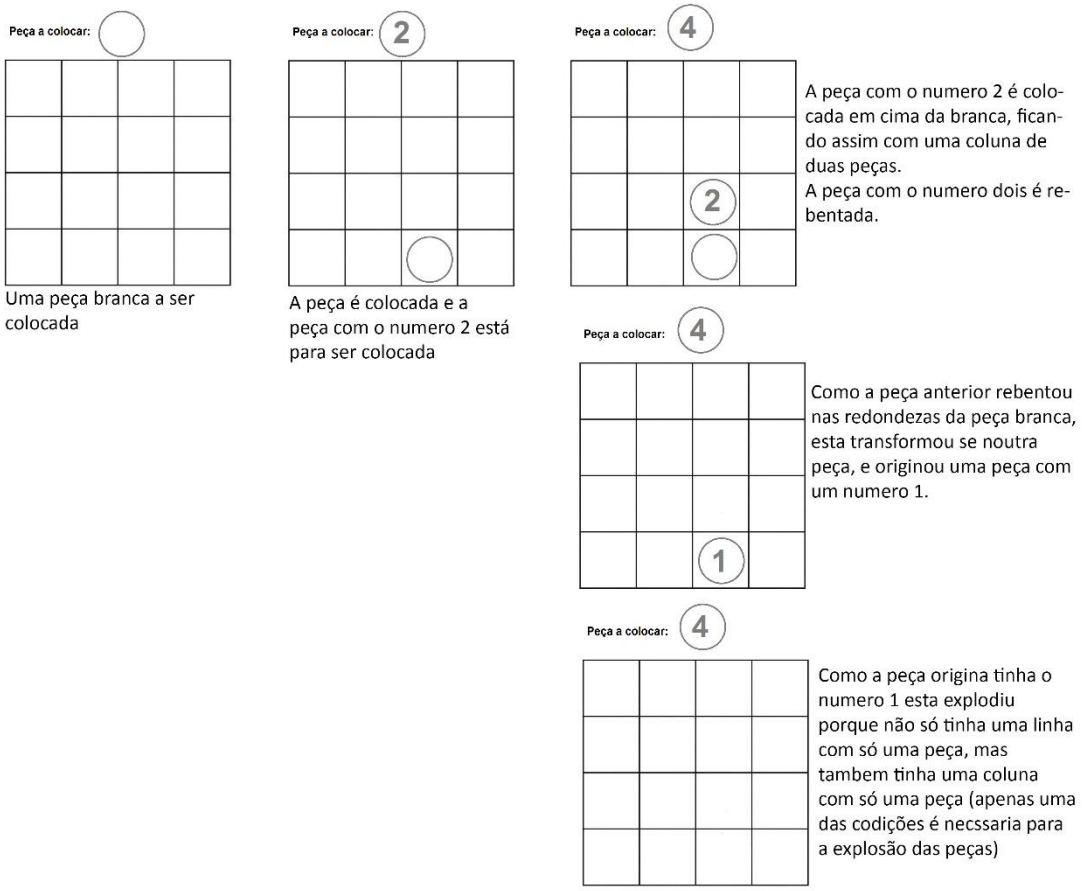

Figura 2 Exemplo do jogo 2

Q-Learning

Q-learning é uma técnica de *reinforcement learning*. *Reinforcement learning* é uma área da inteligência artificial que se baseia em aprendizagem por repetição, ou seja, existe um agente que joga o jogo e vai aprendendo quais as melhores jogadas para cada estado do jogo. Q-learning é baseado nos seguintes conceitos:

- Agent (agente) – é o que interage com o jogo.
- State (estado) – isto representa o estado atual do jogo, no nosso jogo, este estado é representado por 16 casas que representam as posições no tabuleiro 4x4 e a próxima peça a ser jogada.
- Action (ação) – estas são as ações que podem ser tomadas pelo agente em cada estado, ou seja, os controlos do jogo. No nosso caso, temos 4 ações que representam a coluna em que a peça vai ser jogada.
- Reward (prémio) – é os pontos que recebemos, positivos ou negativos, por fazer uma ação num determinado estado. No nosso jogo, o reward representa o numero de peças rebentadas, caso a ação tomada faça com que percamos o jogo o reward é -1.
- Q-table – Isto é uma tabela onde as linhas são os estados e as colunas são as ações, e em cada posição está o Q-value.

$$
\begin{array}{cc} & \textbf{Action} \\ \textbf{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}
$$

Figura 3 Q-table

- Q-value – é o valor para uma ação num determinado estado. A melhor ação é aquela que tem o maior Q-value. Este valor é calculado utilizando a seguinte formula:

$$
Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)
$$

Figura 4 Formula Q-value

Pelo estudo da formula podemos verificar que o Q-value é o valor do reward recebido pela ação mais o máximo dos Q-values do próximo estado, sendo este máximo multiplicado pelo discount factor que representa a importâncias dos futuros rewads.

Entendo todos estes conceitos podemos então demostrar os passos para o funcionamento do Q-learning, para o nosso jogo:

1. Inicialização de uma Q-table em que os Q-values estão todos a zero.
2. Iniciar um jogo drop 4
3. O agente tem uma probabilidade de 80% de escolher a melhor ação e 20% de escolher uma ação aleatória. No inicio, sendo que a tabela esta inicializada a zeros é escolhida sempre uma ação aleatória.
4. A ação é aplicada no jogo.
5. Apartir do reward recebido do jogo é calculado o Q-value para a ação tomada.
6. Voltar ao ponto 3, até a aprendizagem estar completa.

Se seguirmos os passos anteriores a Q-table no final da aprendizagem irá estar completa com Q-values que representam a melhor ação para cada estado do jogo.

Poupança de memoria

Numero de estados possíveis:

$$5^{16} * 5 * 0.1 \approx 76\,000\ milhões$$

Sendo 16 o numero de posições na tabela 4x4, sendo 5 o numero de peças possíveis e o ultimo cinco representa o numero de possibilidades para a próxima peça. A maior parte desses estados não são possíveis no jogo, por isso decidimos multiplicar por 0.1.

Se guardássemos cada estado com um array de inteiros de tamanho de 17 e os Q-values desse estado num array de inteiros de tamanho 4 teríamos 21 inteiros que representariam 21*4 = 84 bytes. Se multiplicarmos a memoria utilizada por um estado pelo numero de estados possíveis obtemos a memoria utilizada pela Q-table:

$$84\ bytes * 7.63 * 10^{10} estados \approx 6.41 * 10^{12}\ bytes = 6\,410\ GB$$

Tento este problema de memoria em mente arranjamos algumas formas alternativas de guardar os dados para que não ocupassem tanto espaço em memoria. A primeira formula de poupar espaço foi guardar o valor do estado num *long* em vez de um *array[17*].

0	0	1	3	0	0	3	3	0	0	0	0	0	0	4	2	4

Este array passa a ser representado pelo seguinte *long*: 00130033000000424. Desta maneira o estado passa de 68 bytes para 8 bytes.

A segunda formula que utilizamos para poupar espaço foi guardar os Q-values em bytes em vez de inteiros, desta maneira passamos a utilizar apelas 4 bytes em vez de 16 bytes. Como o numero máximo de um byte é apenas 127 decidimos aproveitar os valores negativos e em vez de inicializar os Q-values a 0 inicializamo-los a -127, quando são feitos cálculos com os valores adicionamos 127.

Com estas duas formulas passamos de 84 bytes por estado para apenas 12 bytes.

$$12\ bytes * 7.63 * 10^{10} estados \approx 9.16 * 10^{11}\ bytes = 916\ GB$$

Apesar deste numero continuar a ser demasiado grande, é o mínimo que conseguimos. A razão pela qual este numero continua a ser grande é devido à grande quantidade de estados.

Conclusão

Apos a conclusão deste trabalho apercebemos nos que deviríamos ter escolhido um jogo com menos numero de estados possíveis. Foi por causa o numero de estados possíveis que reduzimos o tamanho do tabuleiro de 7x7 para 4x4. Apesar disto e das otimizações feitas a quantidade de memoria que está a ser utilizada continua a ser muito grande.

Apesar destas limitações achamos que os bots que criamos utilizando Q-learning estão bastante bons. Para demostrar os resultados obtidos corremos mil jogos em três bots de diferentes níveis de aprendizagem:

	Numero de passos de aprendizagem	**Tempo de aprendizagem**	**Media de pontos**
botWithoutLearning	0	0	51
smallBot	10 milhões	12 segundos	750
bigBot	1 500 milhões	16 minutos	2 280

Como podemos ver nos resultados obtidos, o numero de passos feitos na aprendizagem tem um grande impacto nos resultados obtidos. Sento quantos mais passos investidos na aprendizagem, mais inteligente fica o bot e desta forma mais pontos são obtidos. No entanto derivado aos problemas memoria, já anteriormente referidos, não foi possível passar de 1 500 milhões de passos na aprendizagem. Quem sabe se ao aumentar ainda mais o numero de passos, obteríamos ainda melhores resultados.

Referencias

1. https://play.google.com/store/apps/details?id=com.zynga.drop7&hl=pt_PT
2. https://en.wikipedia.org/wiki/Q-learning
3. https://deepmind.com/research/dqn/
4. https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0
5. https://www.youtube.com/watch?v=A5eihauRQvo