Aprendizagem por Reforço em Jogo do Tipo Solitário

Relatório Final

Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Elementos do Grupo:

Diogo Mendes – 201605360

Diogo Sousa – 201706409

28 de maio de 2020

Índice

1. Objetivo

Este projeto foi desenvolvido no âmbito da cadeira de Inteligência Artificial e tem como objetivo a aplicação de algoritmos de aprendizagem por reforço num jogo do tipo solitário, neste caso o BoxWorld 2. Pretende-se, com isto, treinar um agente de forma a resolver os níveis do jogo no menor número de movimentos possíveis.

2. Especificação

2.1. Análise do tema

Como mencionado anteriormente, este projeto consiste na aplicação de algoritmos de aprendizagem por reforço num jogo do tipo solitário, neste caso o BoxWorld 2, no contexto do problema.

O BoxWorld 2 consiste num jogo de puzzle em que o jogador controla uma personagem num mundo 2D visto de cima com o objetivo de chegar ao fim de cada nível. No entanto para fazer isso tem de movimentar certas caixas para poder ter um caminho a percorrer. A personagem pode empurrar as caixas para a posição oposta da sua posição desde que o espaço esteja vazio. O objetivo é completar cada nível com o mínimo de movimentos e 'empurrões' possível.

É importante mencionar que, devido a problemas encontrados no desenvolvimento do projeto, foi preciso simplificar bastante o jogo, levando a que todos os níveis criados e utilizados não tenham caixas, sendo só necessário o jogador dirigir-se a saída diretamente para passar os níveis.

2.2. Ilustração de cenários

Decidimos criar sete níveis do jogo, na forma de matrizes 8 por 8, onde cada matriz corresponde a um array 2D. Cada posição no array representa algo diferente: o jogador ('J'), a saída do nível ('S'), uma caixa ('C'), uma parede ('P') ou um espaço vazio ('_'). Formulamos quatro funções de movimento (Para mover o jogador para cima, baixo, esquerda ou direita) que são usadas para explorar uma possível solução. Cada movimento tem como objetivo resultar na solução (quando o Jogador se move para a Saída) ou quando não é possível, para um novo array que represente o resultante estado do jogo. Para além disso, o programa tenta chegar à solução com o mínimo de movimentos possível, visto que cada 'passo' do programa também é um passo do jogo, isto torna avaliar a qualidade de uma solução fácil.

2.3. Abordagem

Perante o problema, foi decidida a utilização de Algoritmos de Aprendizagem por Reforço, nomeadamente o Q-Learning e o SARSA.

2.3.1. Q-Learning

Nesta primeira fase, é necessário definir um indivíduo pertencente à população. Um indivíduo é tipicamente representado em forma binária, onde cada 0 e 1 é nomeado de alelo. Esta representação permite definir um indivíduo como um cromossoma.

Q-Learning trata-se de um algoritmo de aprendizagem por reforço cujo objetivo é ensinar um agente a decidir que decisão tomar dentro de determinadas circunstâncias. Não requere um modelo do ambiente, e consegue lidar com variados problemas, sem qualquer tipo de adaptação.

Este algoritmo encontra uma política ótima, no sentido que maximiza o valor esperado do total da recompensa dentro de todos os vários passos sucessivos, começando pelo estado atual.

2.3.2. SARSA

A definição deste algoritmo encontra-se escondida no seu nome (*State-Action-Reward-State-Action*), ou seja, a principal função para atualizar o Q-value depende do estado atual do agente "S1", da ação que o agente escolhe tomar "A1", da recompensa que o agente recebe por escolher essa ação, do estado "S2" em que o agente entra depois de tomar essa mesma ação e, finalmente, da próxima ação que o agente escolha tomar "A2" no seu novo estado.

Um agente SARSA interage com o ambiente e atualiza a política baseada nas ações tomadas. O Q-value de um estado-ação é atualizado por um erro, ajustado pela taxa de aprendizagem *alpha*. Os Q-values representam a possível recompensa recebida por tomar uma determinada ação num determinado estado, e ainda desconta uma futura recompensa recebida pela próxima decisão que tenha que tomar no estado seguinte.

Algumas otimizações do Q-Learning podem ser aplicadas ao SARSA.

3. Desenvolvimento

O programa foi desenvolvido em C++, utilizando o Eclipse como IDE, no ambiente de desenvolvimento Windows.

3.1. Estrutura da Aplicação

A aplicação foi elaborada em três ficheiros cpp, dois contendo cada algoritmo de reforço utilizado e outro contendo a implementação do jogo em si, e dos menus.

Por fim, a interface gráfica é simples, expondo claramente cada nível para facilitar a interação do utilizador com o programa, bem como a introdução de dados por parte de quem utiliza o programa.

4. Experiências

Após o desenvolvimento dos algoritmos por completo, foi posto à prova um agente em sete níveis diferentes, múltiplas vezes, de forma a recolher dados sobre a eficiência dos algoritmos desenvolvidos.

Seguem-se algumas notas sobre os níveis usados, que podem ajudar a compreender os seguintes resultados obtidos.

Nível 1: Nível praticamente quase sem paredes, um espaço enorme aberto em que a saída não está bloqueada. Possivelmente por causa disto, é o nível que apresenta das piores taxas de sucesso quando o agente é dado pouco tempo de treino (há muitas diferentes ações que se podem tomar, é preciso muito treino para descobrir as melhores).

Nível 2: Nível que consiste inteiramente numa linha definida do início até à saída, com paredes a impedir o jogador de afastar-se desta linha. Como este nível é bastante fácil e só tem uma solução, é de esperar que o número de movimentos ideal seja atingido com pouco treino, e que a taxa de sucesso seja alta também.

Nível 3: Nível bastante parecido com o 2, só que tem um pequeno espaço aberto que permite uma pequena mudança no trajeto direto até à saída. Esperam-se bons resultados como no nível 2, com a possibilidade de serem ligeiramente piores com pouco treino.

Nível 4: Nível parecido com o 1, só que desta vez tem um caminho em linha reta para a saída, e o espaço aberto é mais pequeno. Não deve haver grandes problemas, mesmo com poucos episódios, para passar o nível, mas é esperado que haja uma otimização dos movimentos à medida que o treino aumenta.

Nível 5: Este nível começa com uma parede do lado esquerdo e direito do jogador, havendo duas possibilidades de começo. No entanto, começando a ir por baixo dá a possibilidade de terminar o nível com menos 2 movimentos comparado com o caminho de cima. À medida que mais treino é feito, é esperada uma gradual diminuição da média de movimentos.

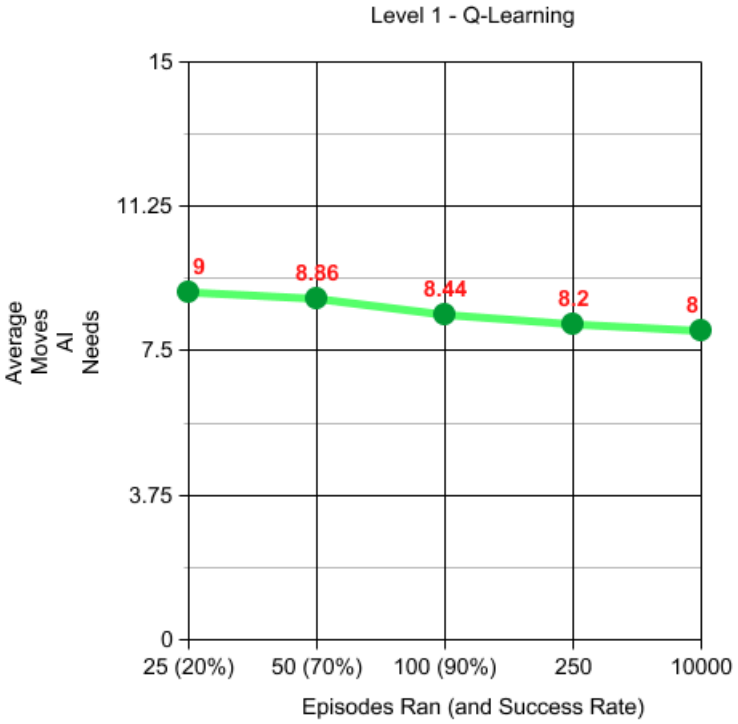
Nível 6: Uma combinação do 1 e do 5, é maioritariamente um espaço aberto mas é dividido principalmente em 2 caminhos principais. É de esperar uma combinação das 2 expetativas de ambos os níveis mencionados.

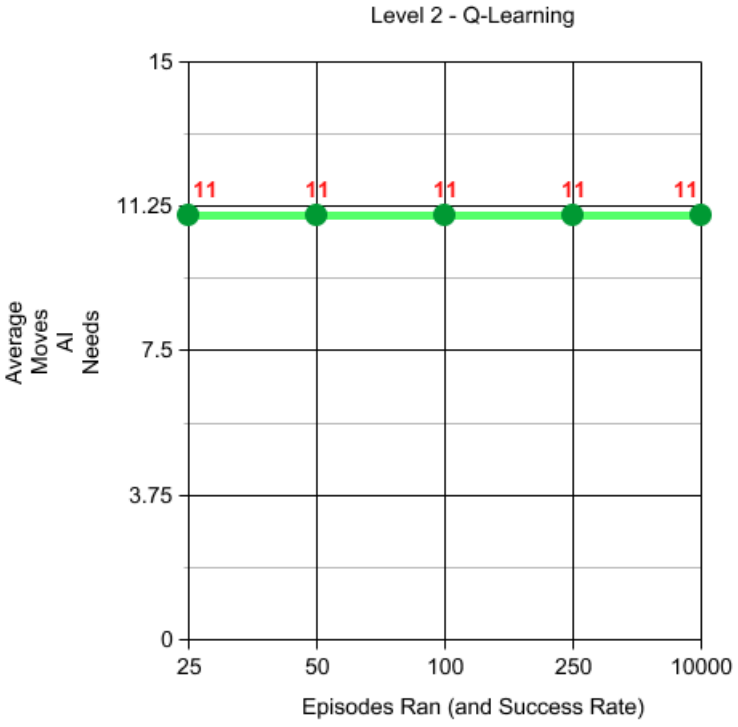
Nível 7: Situação semelhante ao do nível 5, com 2 caminhos iniciais (para cima e para baixo) no entanto neste caso, a diferença entre os dois caminhos não é descoberta imediatamente, só perto do final quando há uma parede que bloqueia a entrada por cima para a saída.

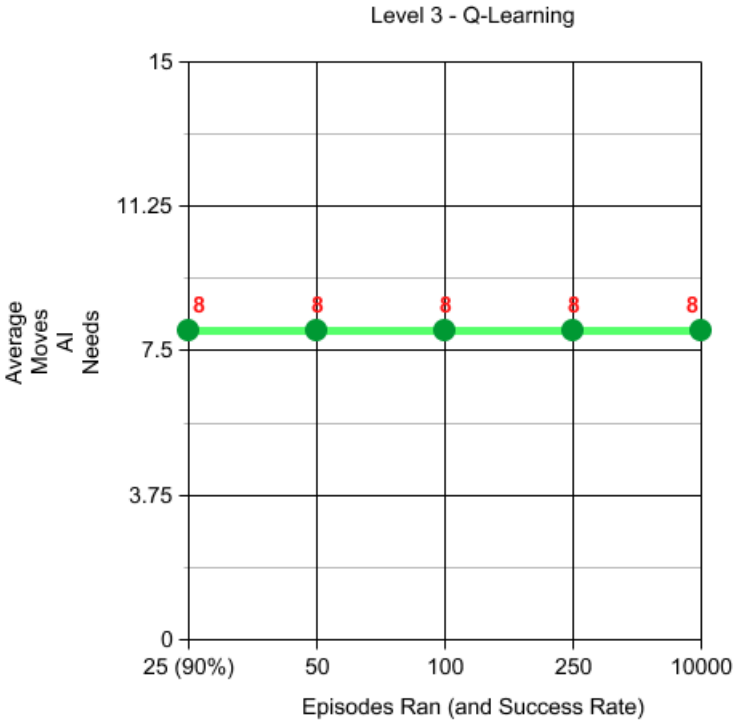
Os testes realizados consistem em treinar um agente um determinado número de episódios num determinado nível, e depois usar a matriz Q obtida do treino para resolver o nível. Cada teste é repetido 10 vezes para cada nível.

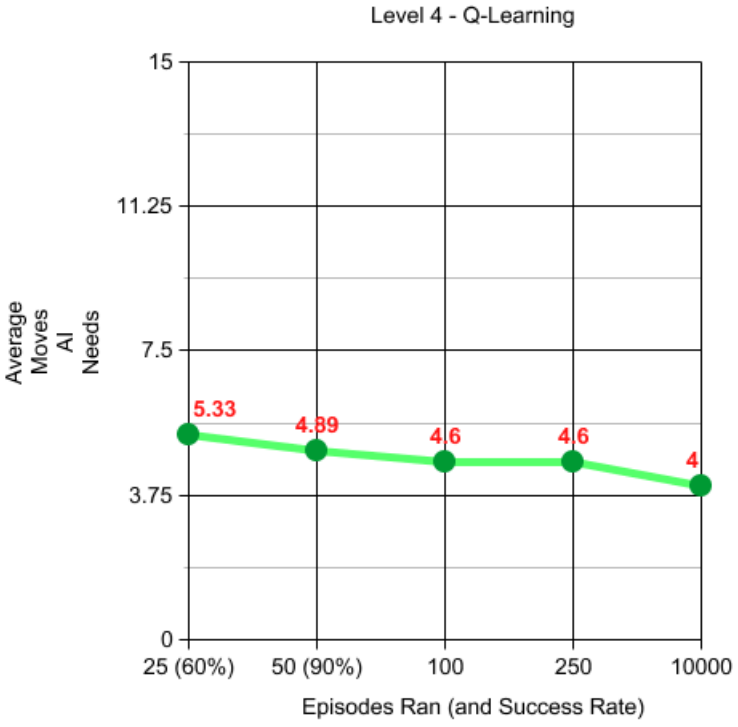
Nos seguintes gráficos o eixo do X representa o número de episódios que o agente treinou, e a taxa de sucesso, isto é, as vezes que o agente consegue chegar ao final do nível com sucesso nos testes (se esta taxa é omitida do gráfico, é 100%). O eixo do Y representa a média de turnos necessários para passar o nível após o treino.

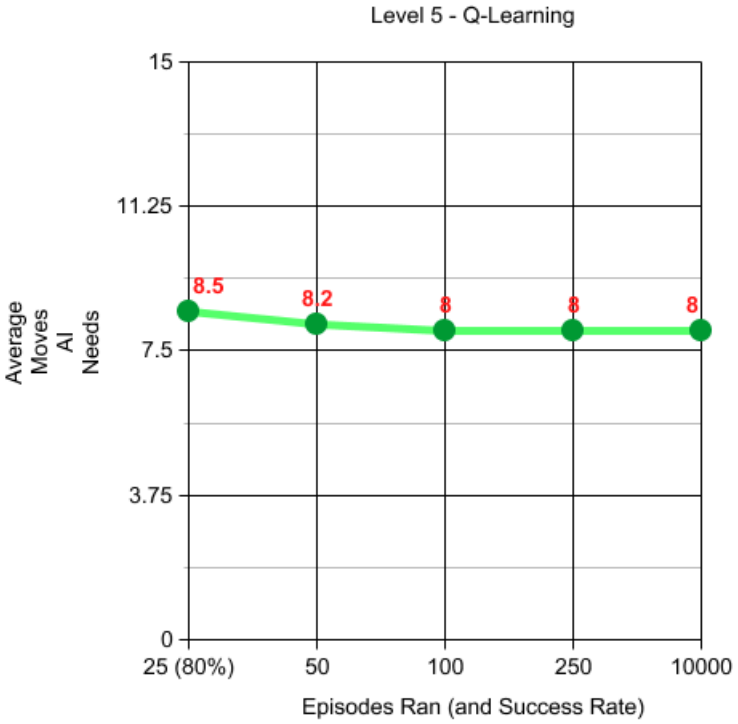
4.1. Q-Learning

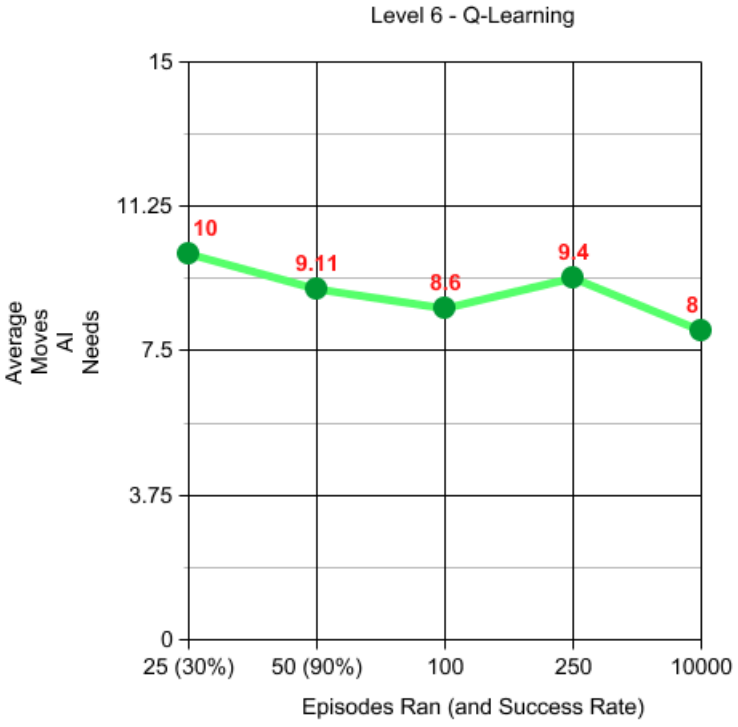

Level 1 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
9
8.86
8.44
8.2
8
25 (20%)
50 (70%)
100 (90%)
250
10000
Episodes Ran (and Success Rate)

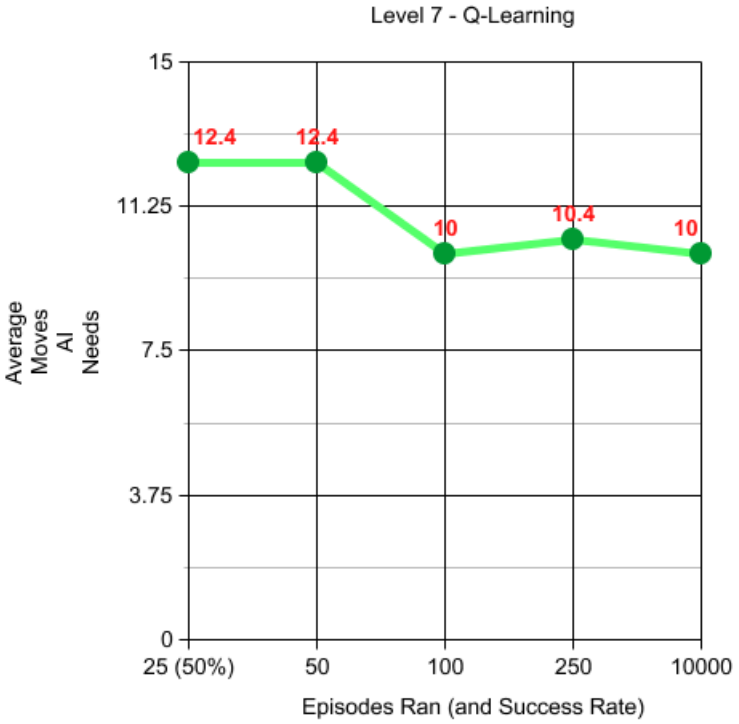

Level 2 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
11
11
11
11
11
25
50
100
250
10000
Episodes Ran (and Success Rate)


Level 3 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
8
8
8
8
8
25 (90%)
50
100
250
10000
Episodes Ran (and Success Rate)

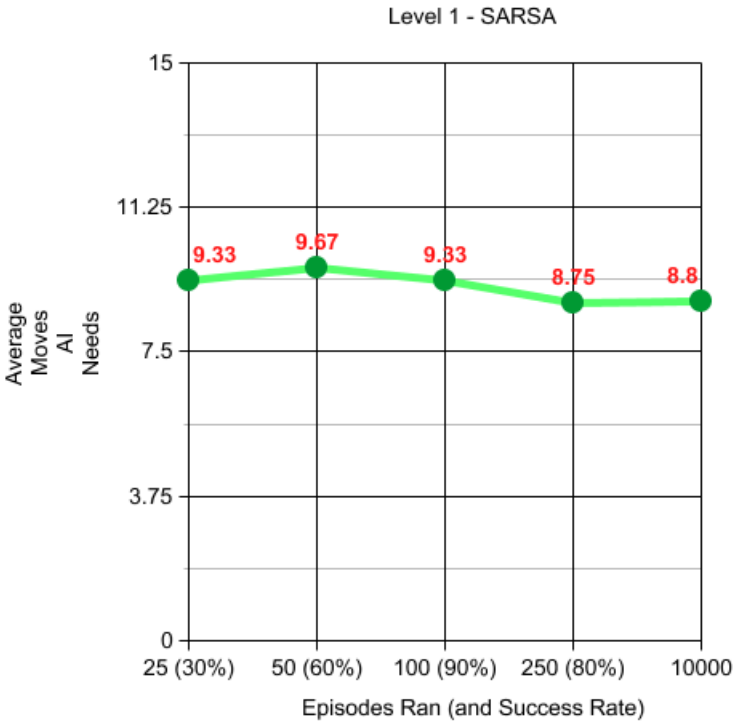

Level 4 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
5.33
4.89
4.6
4.6
4
25 (60%)
50 (90%)
100
250
10000
Episodes Ran (and Success Rate)

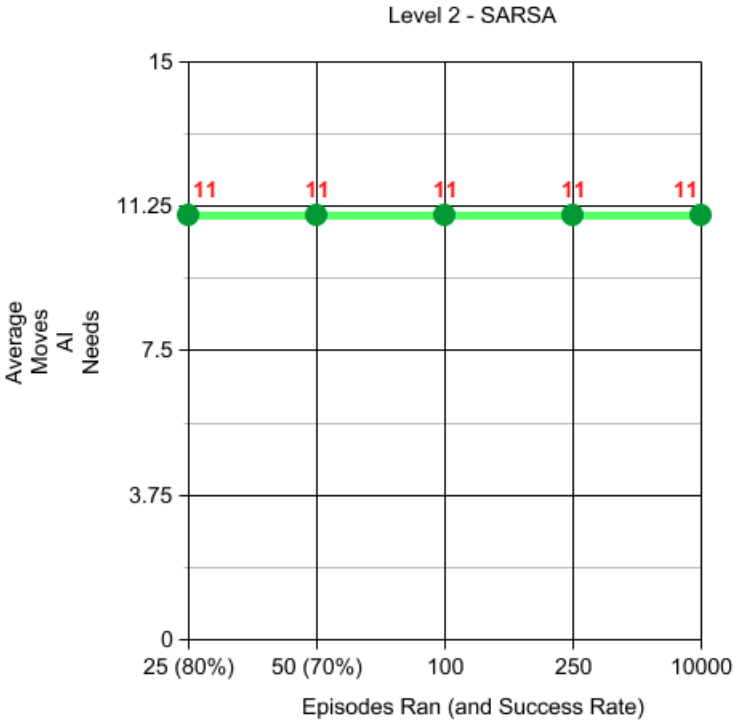

Level 5 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
8.5
8.2
8
8
8
25 (80%)
50
100
250
10000
Episodes Ran (and Success Rate)

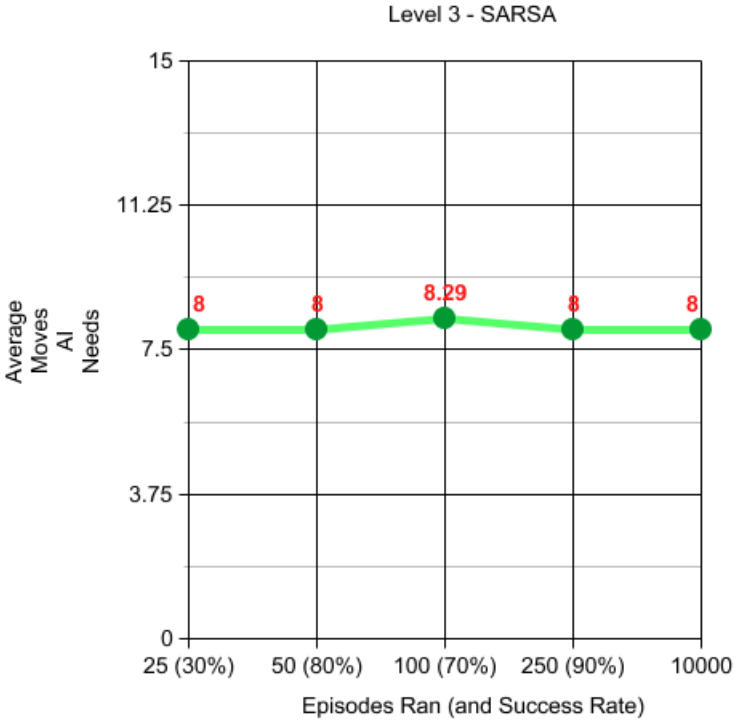

Level 6 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
10
9.11
8.6
9.4
8
25 (30%)
50 (90%)
100
250
10000
Episodes Ran (and Success Rate)

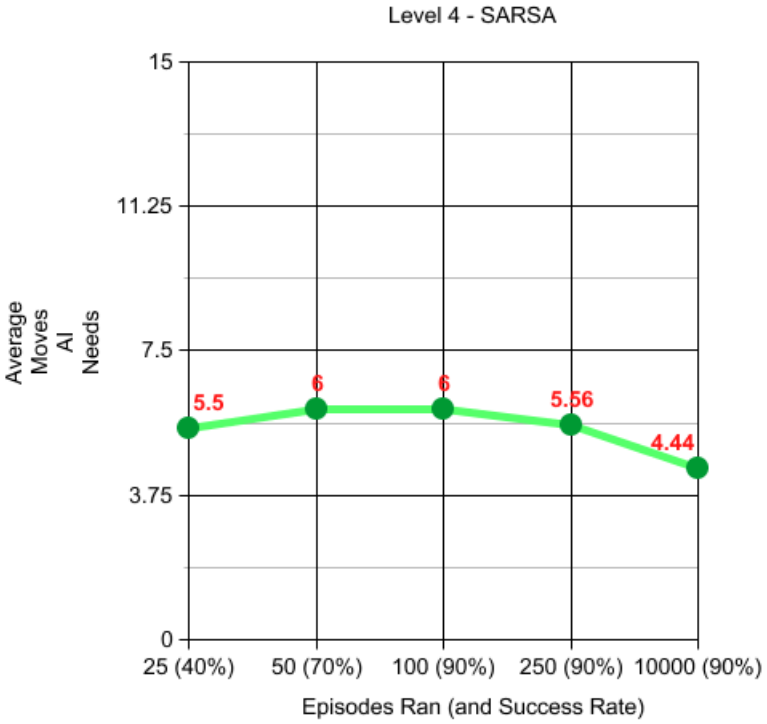

Level 7 - Q-Learning
15
11.25
7.5
3.75
0
Average Moves AI Needs
12.4
12.4
10
10.4
10
25 (50%)
50
100
250
10000
Episodes Ran (and Success Rate)

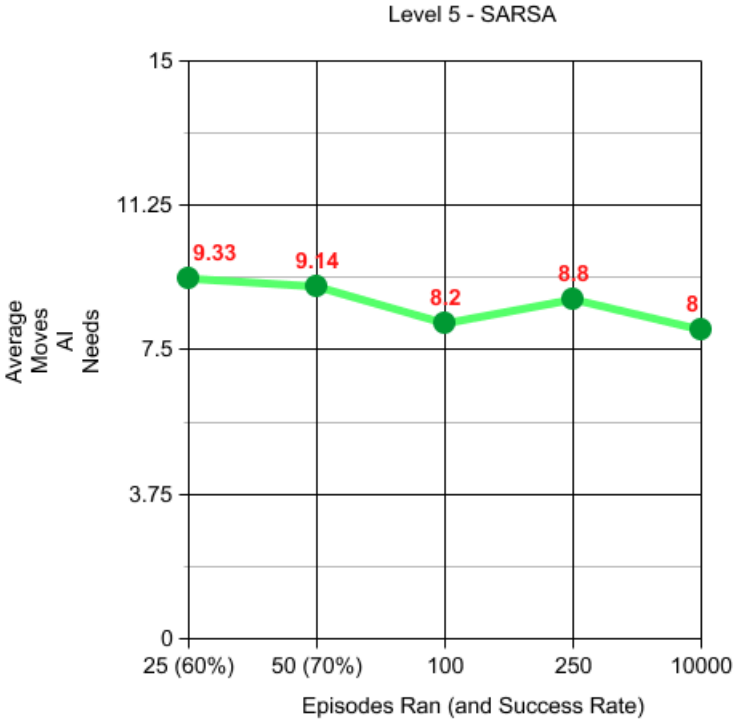
4.2. SARSA

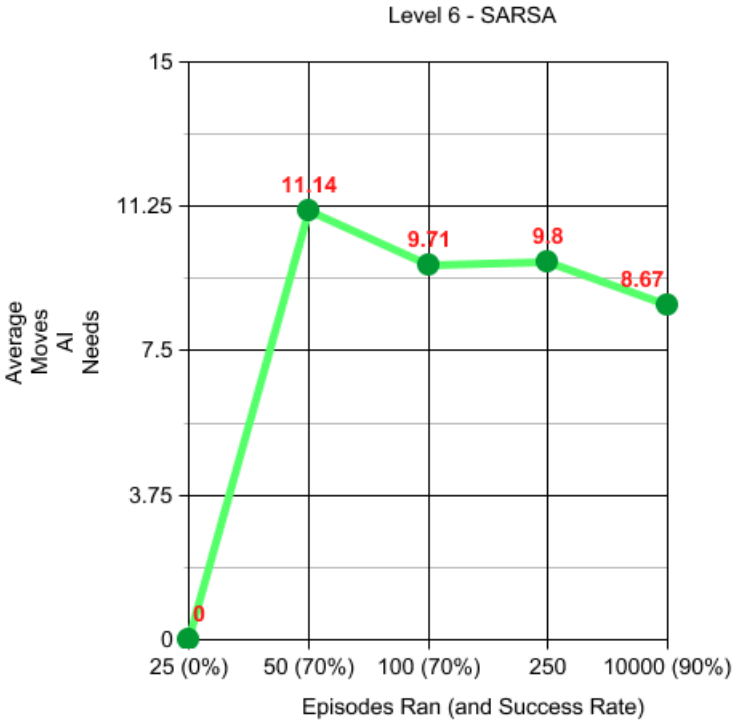

Level 1 - SARSA
15
11.25
7.5
3.75
0
Average Moves AI Needs
9.33
9.67
9.33
8.75
8.8
25 (30%)
50 (60%)
100 (90%)
250 (80%)
10000
Episodes Ran (and Success Rate)

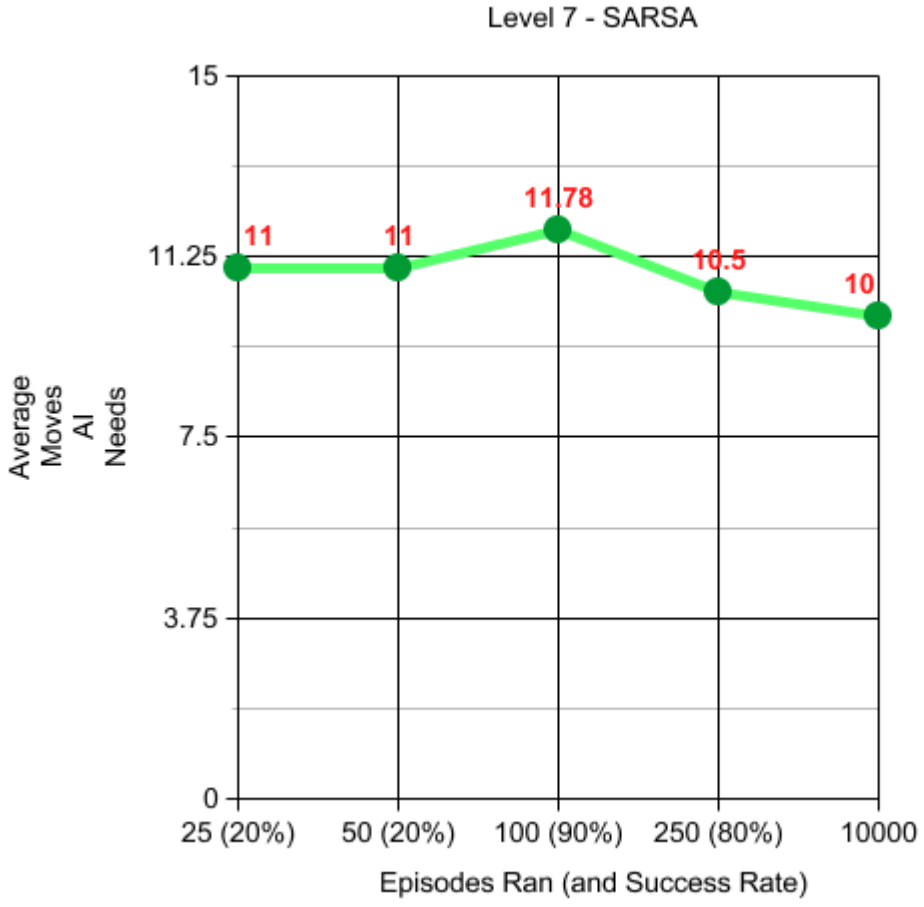

Level 2 - SARSA
15
11.25
7.5
3.75
0
Average Moves AI Needs
11
11
11
11
11
25 (80%)
50 (70%)
100
250
10000
Episodes Ran (and Success Rate)


Level 3 - SARSA
15
11.25
7.5
3.75
0
Average Moves AI Needs
8
8
8.29
8
8
25 (30%)
50 (80%)
100 (70%)
250 (90%)
10000
Episodes Ran (and Success Rate)


Level 4 - SARSA
15
11.25
7.5
3.75
0
Average Moves AI Needs
5.5
6
6
5.56
4.44
25 (40%)
50 (70%)
100 (90%)
250 (90%)
10000 (90%)
Episodes Ran (and Success Rate)


Level 5 - SARSA
15
11.25
7.5
3.75
0
Average Moves AI Needs
9.33
9.14
8.2
8.8
8
25 (60%)
50 (70%)
100
250
10000
Episodes Ran (and Success Rate)





4.3. Análise de Resultados

O algoritmo Q-Learning foi implementado com sucesso, sendo que os resultados obtidos se encontram dentro do esperado. Mesmo com pouco treino a taxa de sucesso do agente a completar o nível não é má, e com mais treino consegue completar sempre os níveis e melhorar a média de movimentos necessários para completar estes mesmos. É de notar até, que com treino de um número alto de episódios (10000 neste caso), o agente consegue resolver todos os níveis de forma perfeita (com o menor número de movimentos possível).

No entanto, os resultados obtidos com o uso do SARSA, não são muito desejáveis. Apesar do agente conseguir, de certa forma, resolver os níveis após o treino, nunca consegue atingir uma taxa de sucesso de 100% em todos os níveis, não consegue resolver todos os níveis perfeitamente mesmo após 10000 episódios, e precisa de mais treino antes de chegar a resultados comparáveis com o Q-Learning. Estes resultados devem-se ao facto de a implementação do SARSA não ter sido feita da melhor maneira. Um ponto a notar é que a maior parte das vezes em que o SARSA não conseguia completar um nível devia-se ao facto que o agente ficava 'preso' num loop infinito de fazer um movimento, e de seguida fazer um movimento que anulava este anterior (por exemplo, estar sempre a ir para a esquerda->direita->esquerda->direita...).

5. Conclusões

O desenvolvimento do projeto foi, em grande parte, despendido na implementação dos algoritmos visto que já tínhamos implementado a estrutura do jogo no primeiro trabalho.

A utilização de algoritmos de aprendizagem por reforço fornece um ganho significativo no que diz respeito ao tempo gasto para se solucionar o problema, pois é possível, em apenas algumas iterações, chegar rapidamente a soluções válidas, formando rapidamente uma solução ideal.

Por conseguinte, os algoritmos de aprendizagem por reforço ajudam sempre a chegar a uma solução cada vez melhor. Num ponto negativo, a solução que obtemos pode não ser a solução ótima, mas não deixa de ser uma boa aproximação.

6. Melhoramentos

A introdução de outros algoritmos de aprendizagem por reforço como PPO ou SAC poderiam vir a tornar o nosso programa mais completo e com soluções ainda mais precisas.

Uma eventual utilização do Unity e ML Agents também seria outra possível melhoria.

7. Recursos

7.1. Bibliografia

1. https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html
2. https://github.com/rhiever/Data-Analysis-and-Machine-Learning-Projects/blob/master/example-data-science-notebook/Example%20Machine%20Learning%20Notebook.ipynb

7.2. Software

1. Eclipse IDE for C++
2. Windows
3. Github

8. Apêndice

8.1. Manual de Utilizador

Para correr o programa, basta importar o projeto para Eclipse ou outro IDE de escolha e correr a partir deste.

No início da execução, será apresentado um menu sobre como executar o programa. Após o preenchimento de certos valores, o programa correrá o algoritmo escolhido e, após terminar, serão mostradas as soluções obtidas.