Aplicação de redes neurais e do algoritmo de Monte Carlo no jogo da cobrinha (Snake)

Erick Soares de Souza, Júlio Henrique Fonseca Rodrigues

[1]Departamento de Informática – Universidade Federal de Viçosa (UFV)
Avenida Peter Henry Rolfs, s/n – 36.570-900 – Viçosa – MG – Brasil

erick.s.souza@ufv.br, julio.h.rodrigues@ufv.br

Abstract. *The Snake game had its first version in 1979 and, later, different variants in computers and in video games. However, even though it has been around for a long time, it's a timeless game, with a certain popularity in current generations. In this context, the work presented in this meta-article was developed to show an approach of different Artificial Intelligence techniques applied in a version of the Snake, in order to gather information about the performance of the agent (the snake itself) during the game and compare, in details, the efficiency of the methods, based on pre-defined criteria.*

Resumo. *O jogo Snake, popularmente conhecido como "jogo da cobrinha", teve sua primeira versão em 1979 e, posteriormente, diferentes variantes em computadores e video games. Entretanto, mesmo sendo de longa data, trata-se de um jogo atemporal, com certa popularidade até mesmo nas gerações atuais. Nesse contexto, o trabalho apresentado nesse meta-artigo foi desenvolvido para mostrar a abordagem de diferentes técnicas de Inteligência Artifical aplicadas em uma versão do Snake, a fim de reunir informações sobre o desempenho do agente (a cobra) durante o jogo e de comparar, detalhadamente, a eficiência dos métodos, com base em critérios pré-definidos.*

1. Objetivos gerais

A versão do jogo da cobrinha desenvolvida neste trabalho respeita as regras originais do Snake, no qual a cobra precisa comer as "comidas" que aparecem na tela para aumentar seu tamanho e sua velocidade, sem que ela encoste em alguma das quatro paredes ou em seu próprio corpo. Porém, a cada comida alcançada, sua cabeça troca de lugar com a ponta da cauda, passando a se movimentar no sentido oposto àquela em que ela se direcionava. Vence o jogo quando a cobra ocupar completamente o mapa. O trabalho visa "treinar" a cobra utilizando IA, de modo a aumentar as chances de vitória, minimizando a quantidade de movimentos realizados por ela para cumprir os objetivos.

2. Trabalhos relacionados

Existem diversos estudos que abrangem o uso de Inteligência Artificial em jogos simples como o Snake. Um dos trabalhos mais influentes realizados pela DeepMind, empresa britânica focada em pesquisas e desenvolvimento de máquinas de inteligência artificial, envolve a utilizaçao de redes neurais (Deep Q-Networks) em uma série de jogos Atari, juntamente com a aplicação de aprendizado por reforço em jogos de tabuleiro, como o AlphaGo [Mnih et al. 2015]. A utilização de princípios de tomadas de decisão e de

otimização de movimento podem ser amplamente aplicadas ao Snake. É válido ressaltar, também, estudos sobre Monte Carlo Tree Search (MCTS) [Browne et al. 2012], que fornecem uma base teórica sólida para seu uso em outros contextos de jogos, e sobre Neuroevolution [Stanley et al. 2019], que combina algoritmos genéricos com redes neurais, contribuindo para a evolução de comportamentos dos agentes.

3. Técnicas e algoritmos aplicados

Redes neurais foram aplicadas ao jogo da cobrinha para criar um agente inteligente (a própria cobra) que aprenda a jogar de maneira eficiente, podendo tomar decisões em tempo real sobre qual direção ir para evitar colisões e vencer o jogo. A rede neural passa por um processo de treinamento, envolvendo técnicas de aprendizado por reforço. Separadamente, aplica-se, também, o algoritmo de Monte Carlo, utilizado como uma técnica de busca e decisão, no qual muitas jogadas são simuladas a partir do estado atual do jogo e, com base nos resultados, é definido o movimento que leva o agente a ter o melhor desempenho no jogo.

3.1. Multilayer Perceptron (MLP) e Deep Q-Learning

Um MLP é uma classe de rede neural feedforward composta por pelo menos três camadas sequencialmente conectadas: uma camada de entrada, uma ou mais camadas ocultas, e uma camada de saída. É uma arquitetura versátil e pode ser aplicada a várias tarefas de aprendizado de máquina, incluindo classificação, regressão e reconhecimento de padrões. Neste trabalho, o PyTorch (biblioteca Python usada para aprendizado de máquina) contribuiu para definir as redes neurais e as funções de perda que usamos para treiná-las. O pacote torch.nn contém um conjunto de módulos para representar as camadas da rede, no qual cada módulo recebe tensores de entrada e calcula os tensores de saída.

Após longos experimentos, foram definidos 16 neurônios para a camada de entrada, que recebe as características do ambiente e o estado atual do jogo, incluindo informações sobre para qual direção a cobra está se movendo; se há a possibilidade de colisão instantânea pela sua frente, esquerda ou direita; a quantidade de blocos entre sua cabeça e seu corpo (essencial para quando ela atinge um tamanho maior e as colisões com o próprio corpo aumentam); juntamente com a distância de sua cabeça até a comida, a fim de reduzir a quantidade de movimentos feitos pela cobra para alcançá-la.

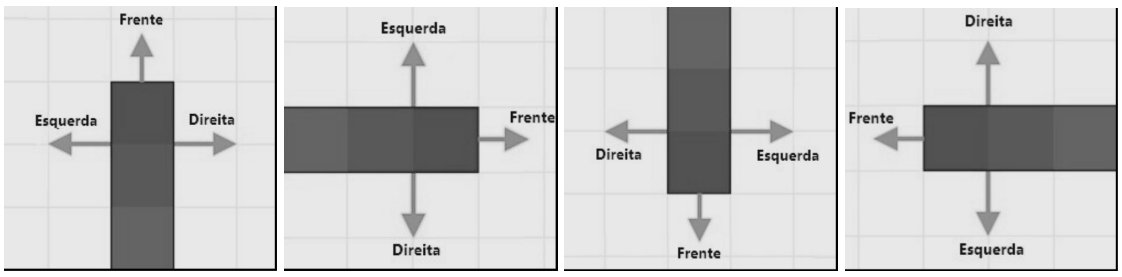

Figura 1. Movimentos do agente no mapa

Para a camada oculta, foram usados 256 neurônios que aplicam a função de ativação ReLu (Rectified Linear Unit) para introduzir a não-linearidade ao modelo, permitindo que ele aprenda uma gama maior de funções de mapeamento entre os estados e

as ações, reconhecendo, também, padrões de comportamentos que levam ao sucesso, ou seja, a maiores pontuações, como evitar paredes e pegar a comida, por exemplo.

Por fim, foram usados apenas 3 neurônios para a camada de saída, representando as possíveis ações que o agente pode tomar (se mover para frente, direita ou esquerda). Os valores gerados para cada neurônio são as Q-values (valores Q) estimadas para cada ação, indicando a expectativa da recompensa que a cobra receberia ao executá-la a partir do estado atual. Nesse sentido, aplica-se o Deep Q-Learning, um algoritmo de aprendizado por reforço, cujo objetivo principal envolve utilizar a rede neural para generalizar valores Q até encontrar uma política otimizada que mapeie as ações ideais que o agente deve tomar para os diferentes estados.

Durante o treinamento da rede, destaca-se também a utilização de uma função de perda e de um otimizador como componentes essenciais para um aprendizado mais preciso. O Erro Quadrático Médio foi a função usada para medir a diferença entre os valores previstos pela rede neural e os valores reais, permitindo que esse erro seja retropropagado e a rede se ajuste para minimizá-lo. Além disso, foi introduzido o Adam, otimizador que ajusta a taxa de aprendizado para cada parâmetro durante o treinamento, ajudando a rede a convergir de maneira rápida e eficaz. A criação de funções que guardam a memória de experiências imediatas e passadas também contribuíram para a eficiência do treino e para a melhora de seu desempenho.

3.2. Monte Carlo Tree Search (MCTS)

O algoritmo de Monte Carlo é uma técnica baseada em simulações aleatórias para resolver problemas complexos. No contexto de aprendizado por reforço e jogos, ele é utilizado para estimar a melhor ação a ser tomada em um determinado estado, realizando diversas simulações e calculando a média dos resultados delas. Analisa-se, então, múltiplos cenários possíveis, calculando suas respectivas pontuações e seguindo a direção que maximiza essa pontuação, aumentando as chances de sucesso no jogo.

Para cada direção, o Monte Carlo simula o desenrolar do jogo várias vezes (definido, neste trabalho, por 40 simulações) para prever o resultado dessa decisão. Durante cada simulação, a cobra se move na direção escolhida e, posteriormente, adota um comportamento aleatório, sempre verificando se a cobra alcançou a comida e contabilizando o número de movimentos realizados, até que não haja mais ações válidas a serem tomadas. A partir desses dados, a pontuação final de cada simulação é calculada utilizando-se a fórmula:

$$(comida + 1) \times (movimentos)^{tamanho\_cobra \times 0.01}$$

Esse cálculo foi resultado de inúmeros testes e foi projetado para valorizar tanto a eficiência em coletar comidas quanto o número de movimentos feitos, ponderado pelo tamanho da cobra. A adição de 1 ao número de comidas assegura que a pontuação não seja zero, enquanto o expoente baseado no tamanho da cobra ajusta a influência do número de movimentos na pontuação. Por último, a direção com a maior pontuação média é selecionada, mas, caso todas as possibilidades tenham a mesma pontuação, a direção é escolhida aleatoriamente.

É válido ressaltar, também, que a complexidade do algoritmo Monte Carlo é dominada pelo número de simulações e de movimentos da cobrinha. Logo, a quantidade de simulações escolhida refletiu a necessidade de equilibrar a precisão das previsões da

técnica com o tempo de execução. Um valor maior que este aumentaria a exatidão, porém prolongaria o período de processamento para tomar uma decisão e exigiria um grande custo computacional. Alguns outros métodos foram aplicados na tentativa de reduzir esse custo, impondo-se limites ao número de movimentos que a cobra pode fazer com base em seu tamanho e na largura do grid, evitando que a simulação continue por um tempo excessivo em um cenário controlado.

4. Resultados e comparações finais

A implantação de redes neurais e de ténicas de aprendizado por reforço no jogo da cobrinha tinha como objetivo principal treinar o agente, de forma a obter maiores chances de vitória, com um número mínimo de movimentos. Porém, ao longo dos experimentos realizados, percebeu-se a dificuldade dos métodos aplicados em garantir que a cobra alcance seu tamanho máximo e vença as partidas, optando pela tentativa de conciliar um tamanho considerável da cobra com um número razoável de movimentos.

Inicialmente, durante o treinamento da rede neural, foram testados diferentes hiperparâmetros na intenção de se ter um maior desempenho do agente no jogo. Inclusive, foi analisada a opção de se utilizar coordenadas polares, informando para a rede neural não só a distância até a comida, mas também o ângulo em que ela se encontrava em relação à cobra. Assim, a camada de entrada possuiria 17 neurônios e o agente conseguiria alcançar a comida de modo mais rápido e com menos movimentos. No entanto, os resultados não foram como esperados, provavelmente por agregrar uma complexidade desnecessária para a rede e por se tratar de uma redundância/ruído, em vista que a posição relativa da comida já estava sendo capturada por outras entradas. Logo, não houve uma melhora significativa para mantermos esse neurônio ativo.

Uma alternativa adotada que demonstrou bons resultados foi a aplicação de epsilon-greedy. Esse método permite ajustar a taxa de exploração à medida que o número de jogos aumenta, ou seja, no ínicio do treinamento, o agente explora bastante, porém, ao longo das partidas, o valor de epsilon é reduzido e a cobra passa a ter um comportamento baseado nas ações que ela aprendeu serem melhores.

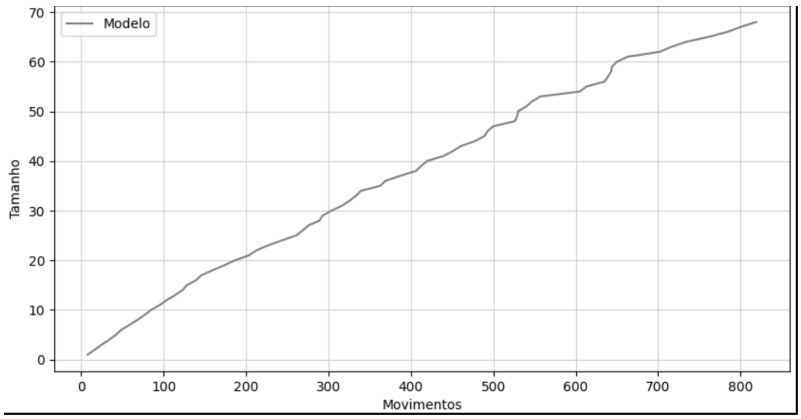

Figura 2. Média de movimentos em 5 jogos com Deep Q-Learning

O gráfico acima retrata a média dos movimentos realizados pela cobra em 5 jogos até atingir um tamanho por volta de 70. O score máximo atingido pelo agente durante o treino foi 96, mas não apresentou constância, obtendo grande variação de tamanho em

todas as partidas jogadas. Visando resultados ainda melhores, foram utilizadas estratégias na maneira como eram distribuídas as recompensas e as penalidades, tanto alterando seus valores quanto abrangendo mais ações indesejáveis da cobra, como ficar em um loop de mesmos movimentos e passar muito tempo sem se alimentar. Entretanto, apesar de atingir um tamanho consideravelmente bom em um número menor de jogos, seu desempenho se manteve estagnado durante a maior parte de seu treinamento. Portanto, essas mudanças não foram mantidas.

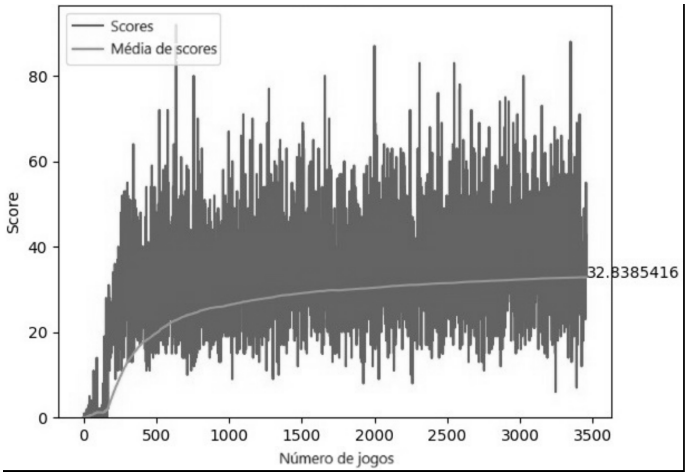

Figura 3. Média dos scores em relação ao número de jogos realizados durante o treinamento

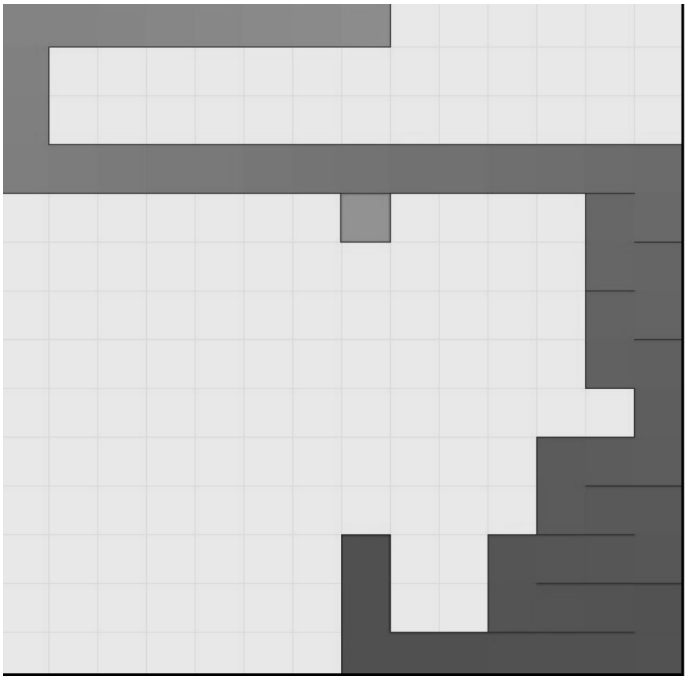

Figura 4. Exemplo de partida jogada com Deep Q-Learning pós-treino

Por outro lado, o algoritmo de Monte Carlo apresentou resultados satisfatórios logo de início. Para tentar melhorar o desempenho da técnica, buscou-se variar e testar o método com diferentes heurísticas e números de simulações a serem feitas. Um dos

principais impasses nesses experimentos foi o tempo de execução do algortimo, já que, como mencionado anteriormente, quanto maior a quantidade de simulações a serem feitas, maior o custo computacional e mais lenta a cobra fica. Desse modo, para cada direção, definiu-se 40 simulações que, apesar de ainda não otimizarem completamente o tempo, garantiram que o agente obtivesse boas pontuações.

Quantos às heurísticas testadas, o critério usado para variá-las foi a influência dos movimentos da cobra na pontuação final de cada partida. Notou-se uma quantidade exacerbada de passos feitos pelo agente para se ter um determinado tamanho. Logo, na intenção de minimizar esse valor, aplicou-se diferentes expoentes no cálculo dos pontos de cada simulação, a fim de encontrar aquele que estimulasse maiores movimentações conforme a cobra crescesse.

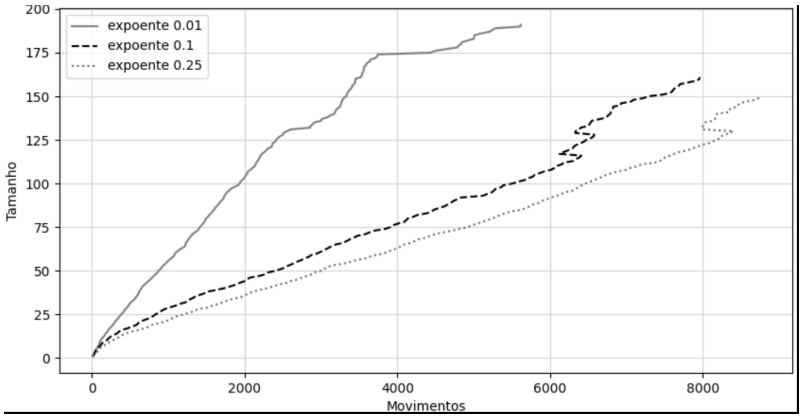

Figura 5. Comparações entre heurísticas com os melhores resultados

A partir do gráfico, é nítido que, utilizando o expoente 0.01 para o cálculo da heurística, o agente atingiu maiores pontuações, reduzindo, também, o número de movimentos praticamente pela metade, se comparado às outras. Assim, foi definida a heurística final, que, felizmente, resultou em um ótimo desempenho da cobra no jogo, chegando bem perto de vencê-lo.

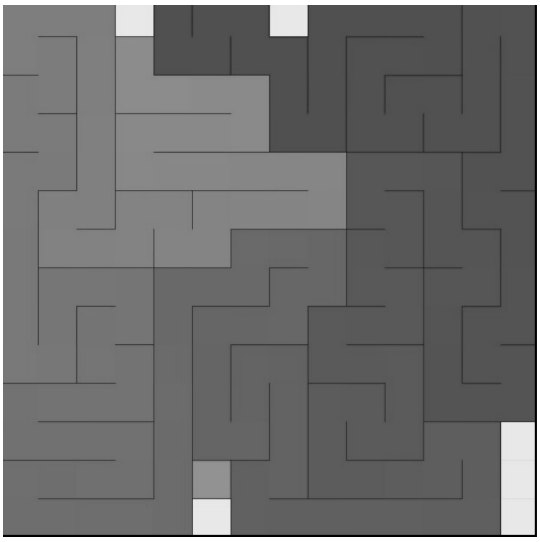

Figura 6. Exemplo de partida jogada com Monte Carlo

Por fim, analisando os resultados das técnicas aplicadas ao jogo, percebe-se uma grande diferença entre o desempenho obtido de ambos, com base nos objetivos definidos nesse trabalho. Apesar de a rede neural alcançar pontuações menores, o número de movimentos feitos pelo agente é significativamente menor se comparado ao Monte Carlo. Este algoritmo, entretanto, garantiu maiores chances de vitória da cobra e de modo mais eficiente, já que simula cenários futuros a partir de estados atuais e não passa por um longo período de treinamento como a rede. Em geral, vale mencionar que, independente das diferenças apontadas, são muito semelhantes no quesito esforço computacional.

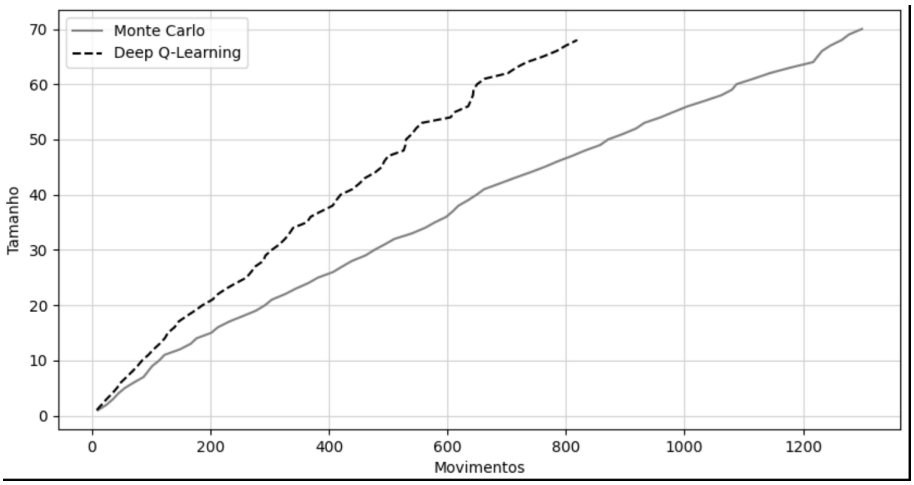

Figura 7. Comparação entre a média de 5 jogos com Deep Q-Learning e com Monte Carlo

Para mais detalhes relacionados à implementação e à aplicação desses métodos, o código encontra-se disponível em https://github.com/ericksoares13/INF420-COBRINHA/tree/master.

5. Conclusão

Em resumo, este trabalho evidenciou que a escolha da técnica de Inteligência Artificial a ser aplicada ao Snake pode variar, baseando-se nos objetivos que buscam ser alcançados. Em relação à rede neural e ao Deep Q-Learning, houveram dificuldades em garantir um desempenho consistente em termos de pontuação máxima, indicando que, embora eficiente em movimentos, esta estratégia de aprendizado por reforço pode ser limitada no quesito exploração e generalização para novos cenários. Já sobre o algoritmo de Monte Carlo, apesar de ir contra a meta de minimizar o número de movimentos, demonstrou-se mais robusto na capacidade de aumentar as chances de vitória do agente.

No entanto, é importante comentar que alguns destes resultados podem não ser válidos para ambientes maiores e mais complexos, levando em consideração que o Monte Carlo, por exemplo, obteria um tempo de execução bem maior que a rede. Portanto, a análise comparativa realizada demonstra que ambas as técnicas têm seus méritos e podem ser complementares em sistemas que buscam equilibrar eficiência e performance em jogos.

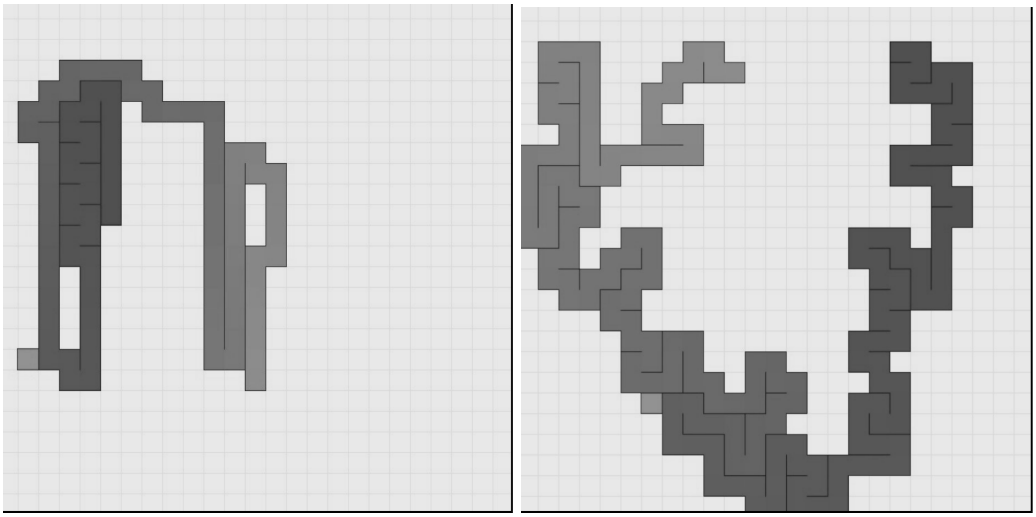

Figura 8. Exemplo de jogo com Deep Q-Learning e com Monte Carlo em um cenário maior, respectivamente

Referências

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., and Ostrovski, G. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.