
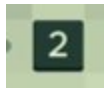


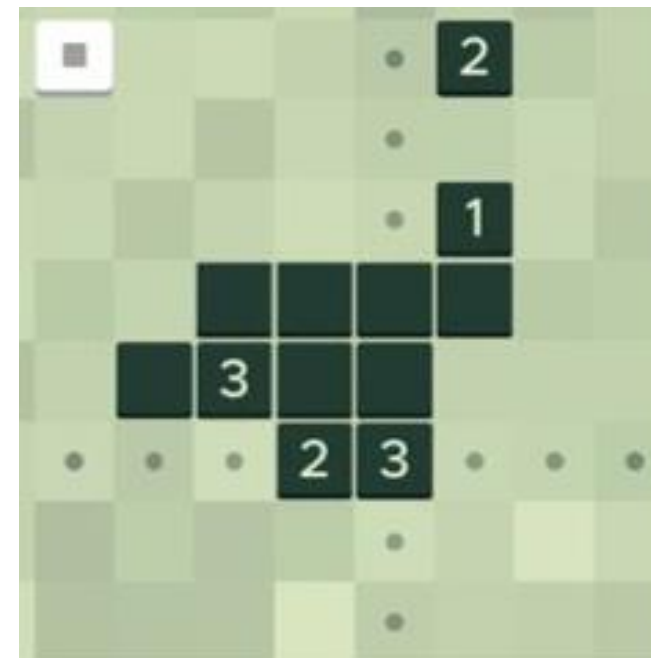
ZHED – Reinforcement Learning

Descrição do Jogo

1. O jogo decorre num tabuleiro quadrangular de tamanho variável.
2. **Objetivo:** preencher o quadrado objetivo 
3. **Como?** Jogando os quadrados numerados 

O número do quadrado representa a quantidade de **casas em branco** (ver figura ao lado) que ele consegue preencher numa de quatro direções: **cima, baixo, esquerda e direita**.

Note-se que quadrados numerados, mesmo antes de serem jogados, já estão a preencher uma casa do tabuleiro.



Em alguns níveis é possível que existam soluções diferentes para o mesmo puzzle.

Referências

Gaming Boulevard page, \url{https://www.gamingboulevard.com/2020/05/indie-corner-zhed/}. Last accessed 20 of May of 2020

Free code camp page, \url{https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/}. Last accessed 24 of May of 2020

Wang, Hanrou, Michael Emmerich, Mike Preuss and Aske Plaat. “Analysis of Hyper-Parameters for Small Games: Iterations or Epochs in Self-Play?” ArXiv abs/2003.05988 (2020): n. pag.

Lee, Jay H.; Shin, Joohyun; Realff, Matthew J., Machine learning: Overview of the recent progresses and implications for the process systems engineering field

Formulação do Problema

O objetivo do trabalho é ensinar um agente a solucionar um puzzle com base em aprendizagem por reforço. Este problema é uma tarefa episódica, dado que começa com todos os quadrados por jogar e chega a um fim natural (estado terminal), onde ou se perdeu, não havendo mais jogadas por fazer, ou se ganhou. Por esta razão escolhemos a abordagem de Monte Carlo para explorar o puzzle nível a nível.

O jogo representa um ambiente **determinístico** e **discreto**, sendo o estado representado pelas células do tabuleiro. Num jogo com N quadrados jogáveis existem $4N$ **ações** possíveis (para cada quadrado jogar numa de 4 direções) e um espaço de **estados** de tamanho $N! * 4^N$.

Os vários hiperparâmetros (learning rate, discount rate, exploration/exploitation ratio) vão ser estipulados de forma a encontrar o melhor processo de aprendizagem para o nosso agente. Contudo, dado que o nosso objetivo é a resolução do jogo, o **gamma** (discount rate) tenderá a ser **elevado** (0.90 a 0.99), logo haverá um desconto menor dado que o nosso agente se preocupa mais com a recompensa a **longo termo** (i.e. chegar ao fim do nível).

No que toca às recompensas, como se pretende atingir o objetivo com o menor número de jogadas possíveis, tencionamos retornar -1 por cada jogada, -10 quando já não mais existem jogadas, $+10$ quando o nível é resolvido.

Algoritmos e Ferramentas

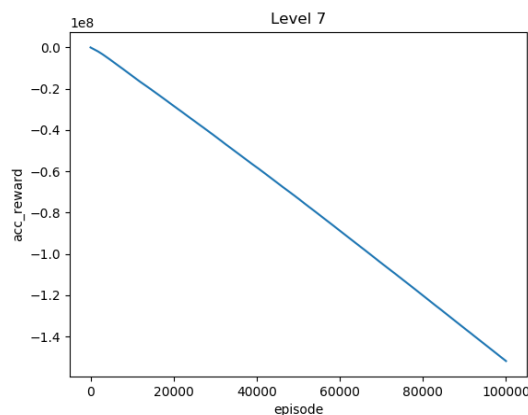
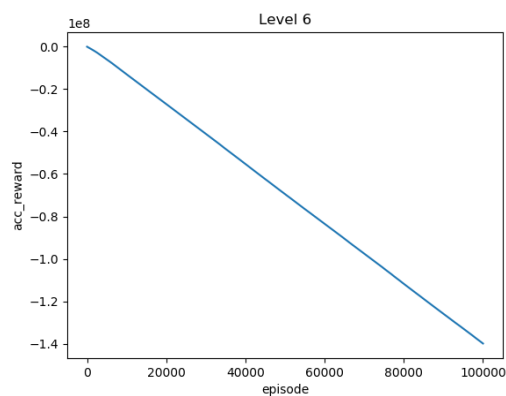
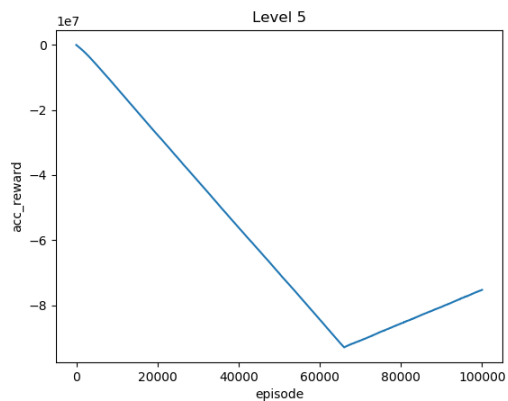
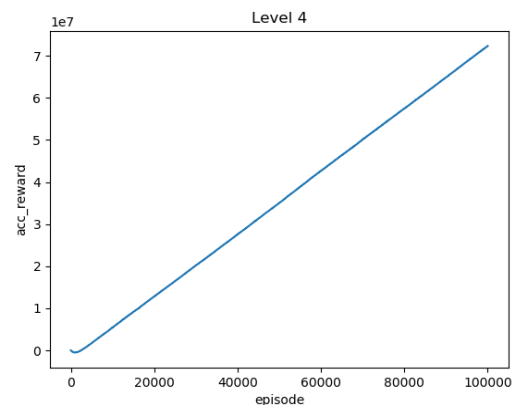
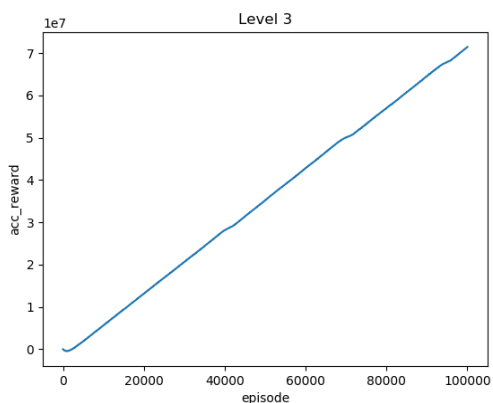
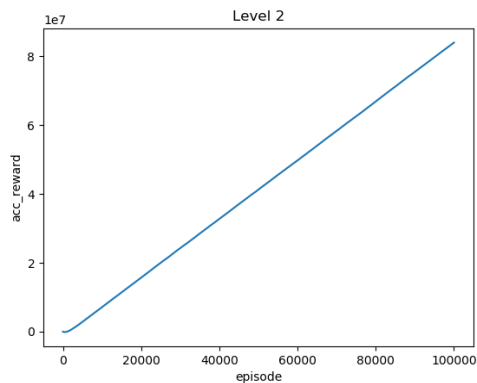
Para o desenvolvimento do algoritmo, tendo em conta que a capacidade de processamento é muito importante, escolhemos a plataforma Google Colab, que nos permite criar um notebook partilhado para desenvolvermos em conjunto a nossa Inteligência Artificial. Para o desenvolvimento do ambiente de jogo decidimos utilizar OpenAI Gym (biblioteca de Python) pela sua simplicidade. Tendo estas ferramentas por base, estamos a considerar os seguintes algoritmos:

- Q-Learning;
- State-Action-Reward-State-Action - SARSA;
- Proximal Policy Optimization – PPO;

Dada a proximidade dos dois primeiros algoritmos apenas foram analisados os resultados obtidos do algoritmo Q-learning e do PPO.

Q-Learning

Comparação das recompensas por nível



Comparação da recompensa acumulada nos primeiros 7 níveis do jogo, demonstrado um evidente ponto de viragem no nível 5.

Os Hiper parâmetros usados foram:

Número de episódios = 100 000

learning rate = 0.01

Max steps = 1000

Gamma = 0.99

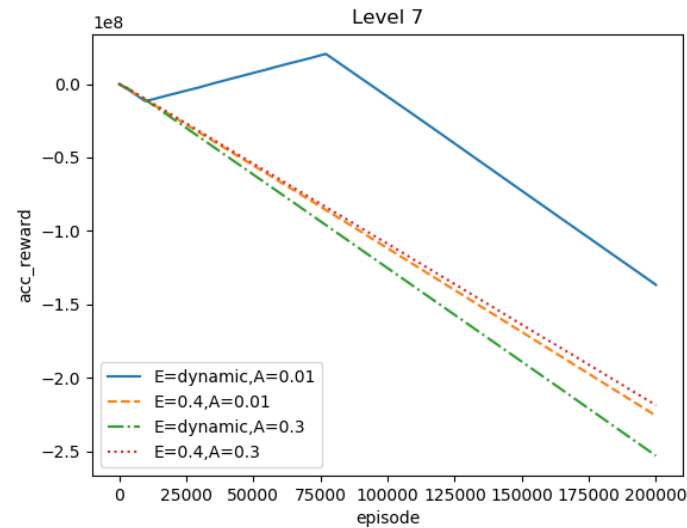
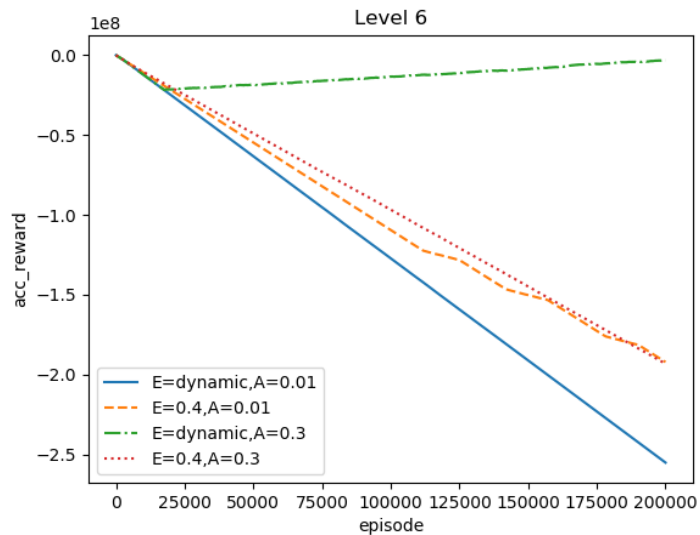
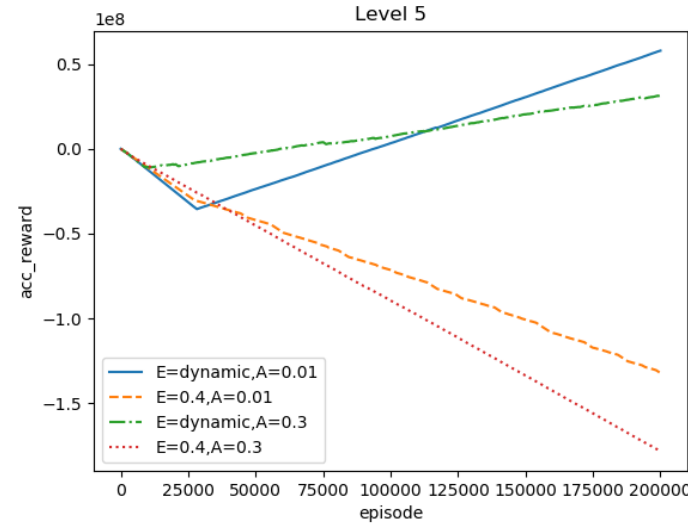
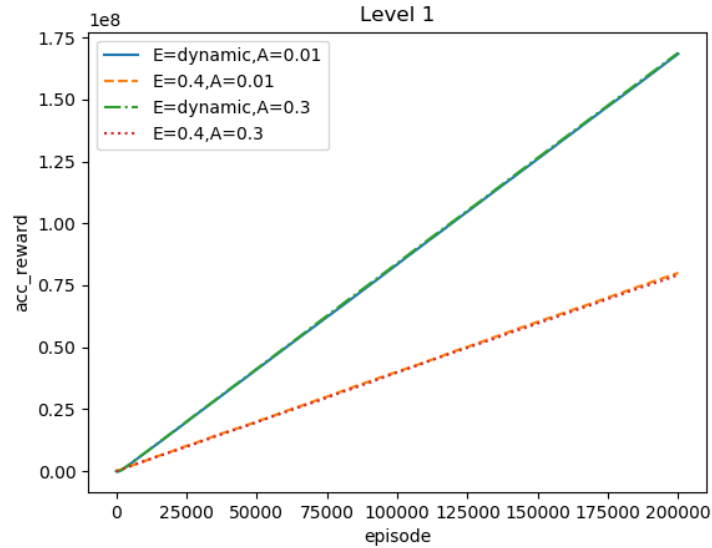
Os parâmetros de exploração usados:

Epsilon = 1.0

Epsilon maximo = 1.0

Epsilon minimo = 0.1

Taxa de decaimento = 0.001



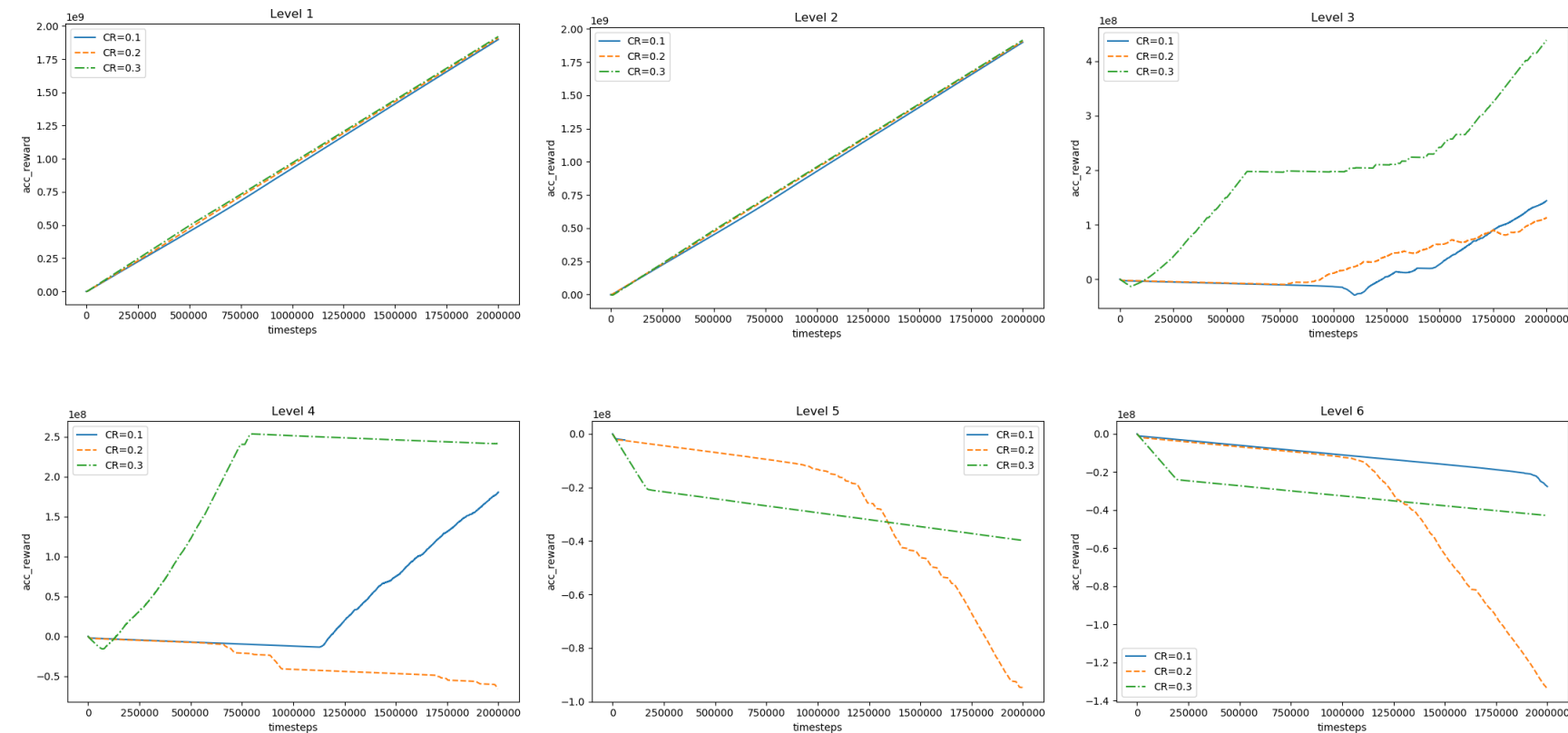
Comparação da variação de hiper parâmetros em diferentes níveis (Q-Learning)

O primeiro nível é bastante simples e por isso não há muito a extrair, mas a partir do nível 5 conseguimos observar que:

- > um epsilon fixo não conduz a resultados favoráveis
- > nos níveis 1, 5 e 7 um alpha maior demonstra a capacidade de começar a produzir resultados mais cedo, mas ao mesmo tempo não tão significativos quanto possivelmente um alpha menor

PPO

Comparação das recompensas por nível



Os níveis mais simplistas 1 e 2 novamente não vêm muita divergência. É a partir do 3 em que se começa a apontar uma considerável melhoria do feedback inicial recebido pelo CR(epsilon) mais elevado seguido mais tarde pelo de 0.2 mediano. O mesmo não se parece observar no de CR menor de 0.1 que nos níveis 3 e 4 estagnava e nos 5 e 6 onde os outros estagnavam começava a piorar cada vez mais. Tendo isto em conta, concluímos o CR maior como sendo o mais eficiente, visto obter os melhores resultados mais rapidamente.

Conclusões

O algoritmo Q-learning conseguiu resolver os puzzles apresentados até ao nível 7. A limitada memória da nossa máquina impossibilitaria o alcance de soluções a partir do nível 12 dado este conter mais de 6 quadrados jogáveis.

O algoritmo Q-learning obteve melhores resultados que o PPO, resolvendo mais níveis e aprendendo mais rápido que este. A combinação de Hiper parâmetros que produziu melhores recompensas acumuladas foi um valor de epsilon variável assim como um valor de alpha dependente da dificuldade do nível a ser resolvido.

Por fim, notámos que Reinforcement Learning é uma forma interessantíssima de Machine Learning, apresentando, contudo, várias limitações.

Num trabalho futuro gostaríamos de comparar os nossos resultados com os de outros tipos de estratégias de entre as várias áreas da Inteligência Artificial, procurando alcançar uma solução que resolva mais níveis e mais eficientemente.