Zhed Puzzle Utilizando Aprendizagem por Reforço

Daniel Brandão
Departamento de Engenharia Informática
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201705812@fe.up.pt

Gaspar Pinheiro
Departamento de Engenharia Informática
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201704700@fe.up.pt

Pedro Moás
Departamento de Engenharia Informática
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201705208@fe.up.pt

***Abstract*— Este artigo contém a descrição de um trabalho no âmbito da unidade curricular de Inteligência Artificial do Mestrado Integrado em Engenharia Informática e Computação na FEUP. O objetivo deste é treinar um modelo de IA para que seja capaz de resolver vários níveis do puzzle Zhed, utilizando algoritmos de Aprendizagem por Reforço. Inicialmente será formulado o problema, seguindo-se uma descrição dos algoritmos utilizados, concluindo-se com a análise de resultados experimentais, comparando-os entre si, tendo em conta fatores como a recompensa obtida, a entropia, percentagem de vitórias e *hit*.**

Palavras-chave*— *Inteligência Artificial, Machine Learning, Aprendizagem por Reforço, ML-Agents, Proximal Policy Optimization, PPO, Soft Actor Critic, SAC

I. Introdução

A Aprendizagem por Reforço é uma área de Aprendizagem Computacional (Machine Learning), focada na forma como um agente toma ações num ambiente, procurando maximizar uma recompensa cumulativa que é atribuída a cada ação efetuada. Neste trabalho, foram aplicados algoritmos desta área, com o objetivo de treinar um modelo de Inteligência Artificial que seja capaz de resolver vários níveis do puzzle Zhed.

II. Descrição do problema

O puzzle Zhed é um jogo de tabuleiro em que para passar cada nível é necessário que uma célula expandida atinja uma célula objetivo. Cada célula pode ser expandida numa de quatro direções, podendo sobrepor outras células. Cada célula expande n células na direção escolhida, decrementando n em um por cada célula vazia atingida. Quando uma célula expandida sobrepõe uma outra não vazia, o valor de n não é decrementado, aumentando em um o valor total de células atingíveis pela célula numerada.

Fig. 1. Exemplo de um nível Zhed.

III. Formulação do problema

Para cada nível do puzzle, as posições das células numeradas, da célula objetivo e o tamanho do tabuleiro variam. Assim, usamos uma matriz (lista de listas) M, de dimensões NxN, sendo N então usado para descrever o tamanho do tabuleiro. Os valores guardados na matriz foram escolhidos de forma a facilitar a manipulação do estado atual da célula, podendo cada uma ter o seu valor, *val*, igual a:

- Um número positivo, representando uma peça com valor *val*, o tamanho expansível da célula.
- 0, representado uma célula vazia.
- -1, representando uma célula expandida.
- -2, representando uma peça de destino.
- -3, representando uma peça de destino alcançada.

Com isto, temos que o estado inicial do tabuleiro consiste numa matriz em que:

- Existe pelo menos uma célula com *val* = -2;
- Existe pelo menos uma célula com *val* > 0;
- Não existe nenhuma célula com *val* = -1 ou *val* = -3.

Por outro lado, o estado final do tabuleiro é representado por uma matriz em que:

- Existe pelo menos uma célula com val = -3.

IV. Trabalho relacionado

No sentido de pesquisar sobre trabalho relacionado com o nosso tema, procuramos projetos que tinham por objetivo aplicar algoritmos de Aprendizagem por Reforço em puzzles como o 2048 [2] ou Tetris [4], [5], assim como outros puzzles [1], [3].

V. Ferramentas

Para implementar o jogo, utilizou-se a plataforma de criação de videojogos Unity3D, com o plugin ML-Agents (versão 1.0.2), que nos permitiu aplicar os algoritmos de Aprendizagem por Reforço.

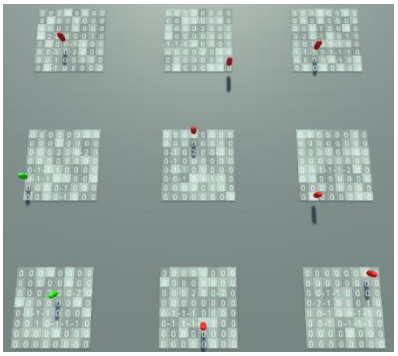

Fig. 2. 9 agentes a tentar resolver alguns níveis de Zhed, simultaneamente.

VI. Algoritmos

Esta secção detalha os algoritmos utilizados no Zhed Solver.

A. *Algoritmos de aprendizagem por reforço*

Neste trabalho, foram implementados os algoritmos disponíveis no ML-agents: Proximal Policy Optimization (PPO) e Soft Actor Critic (SAC). Estes diferem entre si na medida em que o algoritmo SAC é um algoritmo *off-policy*,

sendo que, neste, o computador poderá aprender com as experiências obtidas em qualquer episódio anterior. Em ambos os casos, o programa tenta calcular qual a melhor ação a tomar (*policy*) para um determinado estado observado, obtendo feedback das suas ações através de uma recompensa.

Outro aspeto no qual os algoritmos PPO e SAC se diferenciam é nos hiper-parâmetros disponíveis, sendo possível variá-los de modo a obter melhores (ou piores) resultados. Os mais analisados para o algoritmo PPO foram: λ (lambda), que pode ser visto como o quão o agente valoriza futuras recompensas (visão a longo prazo), ε (epsilon), que influencia o quão rapidamente a *policy* pode evolui, e β (beta), que determina a força da entropia durante o treino. Para o algoritmo SAC, os parâmetros analisados foram: init_entcoef, um regularizador da entropia no início do treino, e τ (tau), que corresponde à magnitude da alteração de Q durante a atualização do modelo SAC.

B. *Abordagem **Wild***

Aplicando os algoritmos ao problema descrito, foram implementadas, essencialmente, duas abordagens. Na primeira, o agente faz uma única observação ao meio, que é a matriz de tabuleiro inteira, tendo de escolher os seguintes valores para a sua ação:

- Coordenadas onde jogar
- Direção da jogada (Cima, baixo, esquerda, direita)

Tanto o espaço de observação como o espaço de ações é, então, discreto. O agente é recompensado se a jogada for válida, mas punido ligeiramente (cerca de 20 vezes menos, em valor absoluto) se não o for, recebendo uma recompensa um pouco maior se o tabuleiro resultante ficar mais próximo de estar resolvido, mas uma punição menor se a jogada for próxima de uma jogada válida. Finalmente, vitórias e derrotas resultam, do mesmo modo, em recompensas positivas e negativas, respetivamente.

C. *Abordagem **Regular***

Na segunda abordagem, o agente efetua as seguintes observações do meio:

- Peças disponíveis para jogar (coordenadas e valor)
- Coordenadas da peça de destino

O agente tem de escolher os seguintes valores para a sua ação:

- Peça a jogar
- Direção da jogada (Cima, baixo, esquerda, direita)

Tal como na abordagem anterior, trata-se, portanto, de um espaço discreto. Neste caso, a recompensa dá muito mais ênfase à qualidade das jogadas, beneficiando aquelas que mais aparentam levar a uma solução. Para isso, foi utilizada uma função de avaliação do tabuleiro que soma a média dos valores de extensão de cada peça, penalizando fortemente os casos em que não há peças alinhadas com a peça de destino. Do mesmo modo, também estão definidas recompensas para vitórias e derrotas.

No geral, o processo de aprendizagem variará bastante, pois, enquanto na abordagem Wild os agentes estarão inicialmente a tentar descobrir quais são as coordenadas certas, e só depois resolver o puzzle, na abordagem Regular o foco estará sempre em resolver o puzzle baseado nas peças que tem à disposição. Em ambas as abordagens, os episódios terminam quando o tabuleiro chega a um estado de derrota ou vitória.

VII. Resultados Experimentais

Durante o desenvolvimento do trabalho, foram realizadas várias experiencias de forma a testar as duas abordagens criadas. Ambos os métodos conseguiram aprender e resolver níveis iniciantes, sendo possível encontrar uma análise dos resultados obtidos abaixo.

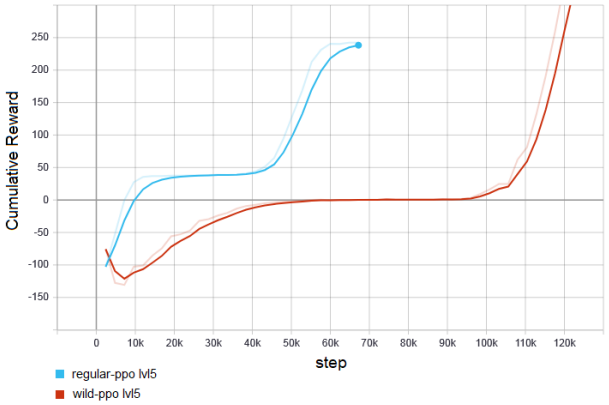

Fig. 3. Recompensas acumuladas em função do número de passos, nas abordagens Regular e Wild, nível 5.

A abordagem Regular consegue rapidamente aprender a resolver um nível graças à função de avaliação robusta. No início, aprende rapidamente a evitar ser penalizado por gastar todas as peças alinhadas com a peça de destino (o que garante que o puzzle fique impossível de completar com sucesso). De seguida, o agente continua a otimizar o estado do tabuleiro até que consegue completar o nível. No caso da abordagem Wild (nota: as abordagens têm diferentes valores para recompensas) o agente aprende primeiro a utilizar as peças dos tabuleiros e, eventualmente, consegue completar o puzzle.

Os agentes foram também treinados, com ambas as abordagens, utilizando cerca de 10 puzzles com semelhante nível de dificuldade. Tanto o Wild como o Regular conseguiram aprender os padrões que levam a completar os níveis, conseguindo posteriormente conseguiram resolver um nível para o qual não foi treinado. (regular com 100% de taxa de sucesso e Wild cerca de 50%)

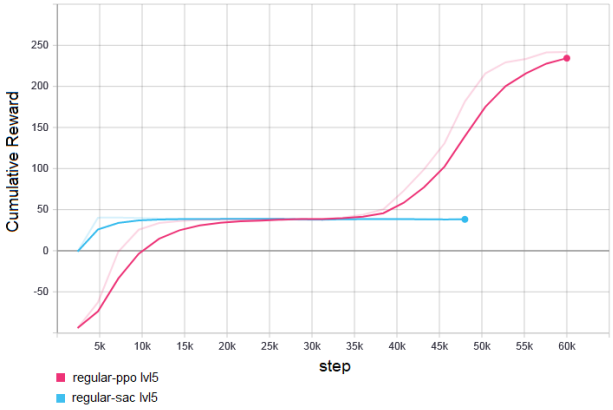

Fig. 4. Recompensas acumuladas dos algoritmos PPO e SAC na abordagem Regular, nível 5.

Á medida que realizámos as experiências apercebemo-nos que o algoritmo SAC não levava a resultados favoráveis para o nosso puzzle, conseguindo raramente completá-lo. Este algoritmo era também mais lento (2 vezes mais lento por cada passo) e exigente para as máquinas. No entanto, para a abordagem Regular, o algoritmo era significativamente mais rápido a evitar ser penalizado, por gastar todas as peças alinhadas.

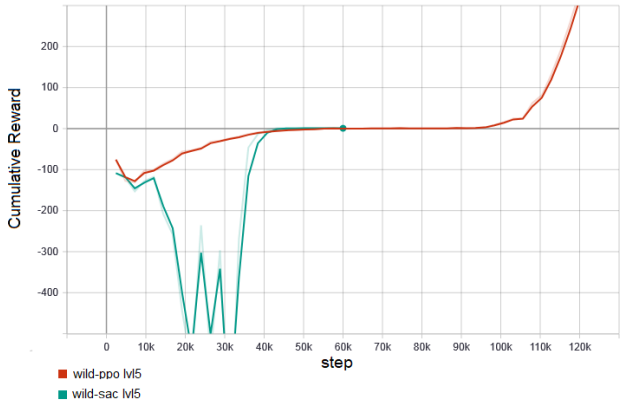

Fig. 5. Recompensas acumuladas dos algoritmos PPO e SAC na abordagem Wild, nível 5.

Na abordagem Wild, os resultados foram ainda mais negativos, havendo um número significativo de picos de recompensas acumuladas, até estabilizar, nunca conseguindo completar o nível.

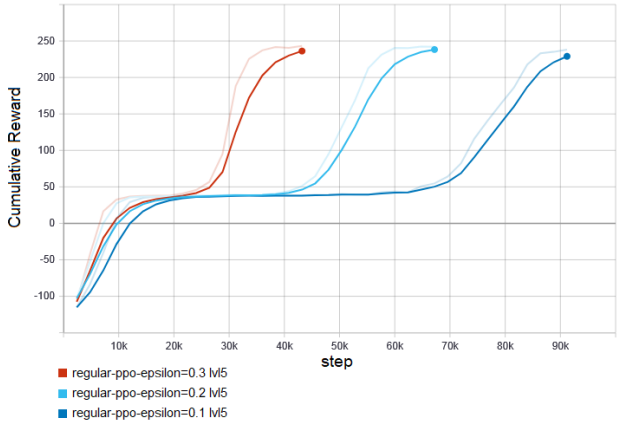

Fig. 6. Recompensa acumulada no algoritmo PPO, abordagem Regular, nível 5, variando epsilon

Como esperámos, o aumento do parâmetro epsilon aumentou significativamente a performance do agente, uma vez que a *policy* pode evoluir rapidamente, sendo muito beneficial para aprender a resolver um nível, uma vez que qualquer incremento na recompensa provavelmente implica estar mais próximo da solução final. Por outro lado, com um epsilon menor a *policy* é mais lenta a evoluir, diminuindo a performance do agente.

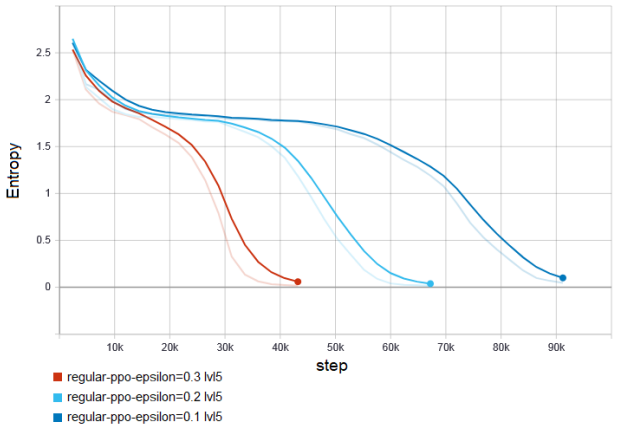

Fig. 7. Entropia no algoritmo PPO, abordagem Regular, nível 5, variando epsilon

É também possível verificar que com o aumento do parâmetro epsilon, a entropia do agente aproxima-se mais rapidamente do 0, visto que, como é mais eficaz a descobrir otimizações no puzzle, é também mais rápido a dar maior probabilidade a certas ações, diminuindo assim a entropia (que é máxima quando todas as ações tem a mesma probabilidade e mínima quando uma ação tem probabilidade de 1).

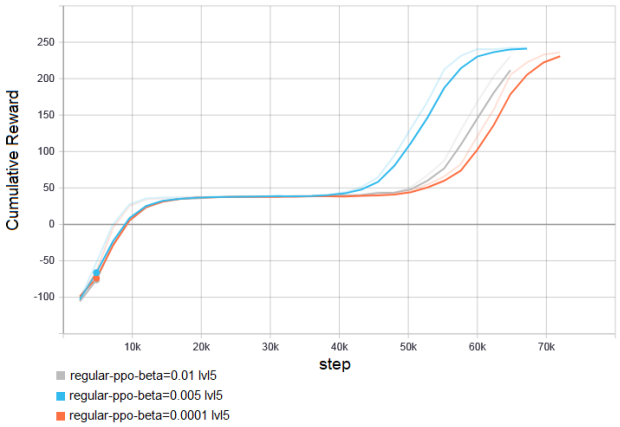

Fig. 8. Recompensa acumulada no algoritmo PPO, abordagem Regular, nível 5, variando beta.

As mudanças no parâmetro beta, que trata da força do regulamento da entropia, não mostraram fazer efeito notável nos resultados do agente.

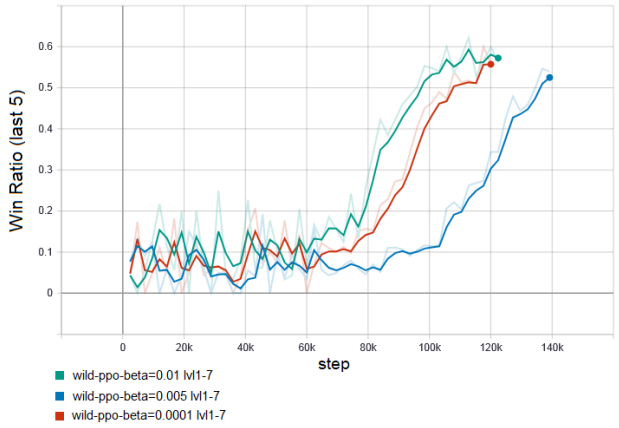

Fig. 9. Percentagem de vitórias no algoritmo PPO, abordagem Wild, níveis 1 a 7, variando beta.

Ao fazer experiências com um agente a tentar resolver múltiplos níveis pudemos verificar que utilizar um beta de valor inferior tem um impacto novamente que o impacto do hiper-parâmetro beta é desprezável no intervalo utilizado, no nosso problema. Neste exemplo podemos observar também, que a abordagem wild, após cerca de 60k steps a jogar os primeiros 7 níveis, a sua win-rate começa a aumentar consideravelmente (até cerca de 55%), implicando que o agente começa a aperceber-se de padrões e técnicas que permitem completar uma ampla gama de níveis.

VIII. Conclusões

Com base nos resultados experimentais acima mencionados, podemos concluir que ambas as abordagens testadas obtiveram resultados positivos. No entanto, notam-se diferenças entre as duas sendo que cada uma apresentou vantagens e desvantagens em relação à outra, em determinados aspetos. Apesar disto, os resultados mostram também que com o algoritmo PPO obtivemos melhores resultados do que com o algoritmo SAC. Gostaríamos também de notar que a ferramenta utilizada, ML-Agents, talvez não seja completamente destinada para resolver este tipo de puzzles.

Dito isto, todo o estudo envolvido levou a um aprofundamento dos conhecimentos relacionados com a aprendizagem por reforço, acabando por ser uma excelente introdução ao trabalho com aprendizagem computacional.

Referências

[1] H. Oonishi, and H. Iima, “Improving Generalization Ability in a Puzzle Game Using Reinforcement Learning”, IEEE’s 2017 Conference on Computational Intelligence in Games. Available: http://www.cig2017.com/wp-content/uploads/2017/08/paper_71.pdf [Accessed: May 24th, 2020]

[2] J. Amar, and A. Dedieu, “Deep Reinforcement Learning for 2048”, Massachusetts Institute of Technology. Available: http://www.mit.edu/~amarj/files/2048.pdf [Accessed: May 24th, 2020]

[3] D. Budakova, and V. Vasilev, “Applying Reinforcement learning to find the logic puzzles solution”, Technical University of Sofia, Plovdiv Branch. Available: https://journals.tu-plovdiv.bg/index.php/journal/article/view/16/14 [Accessed: May 24th, 2020]

[4] M. Stevens, and S. Pradhan, “Playing Tetris with Deep Reinforcement Learning”, Stanford University. Available: http://cs231n.stanford.edu/reports/2016/pdfs/121_Report.pdf [Accessed: May 24th, 2020]

N. Faria, “A deep reinforcement learning bot that plays tetris”, https://github.com/nuno-faria/tetris-a