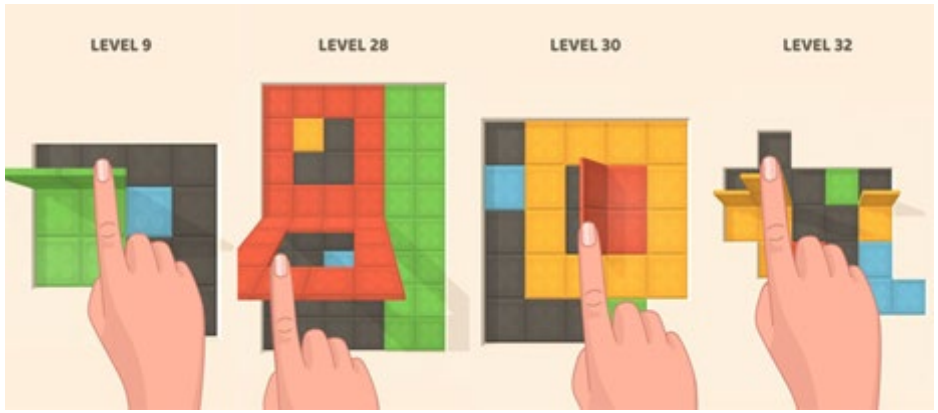
IART – Entrega Final

Alexandre Carqueja – 201705049
Henrique Santos – 201706898
Tiago Alves - 201603820

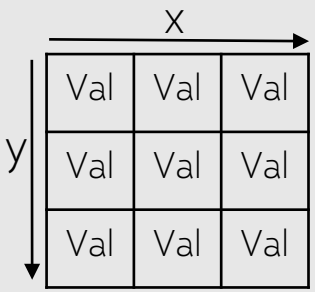


Folding Blocks – SinglePlayer Game

- "Folding Blocks" é um jogo tipo puzzle, em que o objetivo é desdobrar os blocos de modo a preencher todo o tabuleiro
- Quando se desdobra um bloco, este é estendido com as mesmas dimensões na direção do movimento.
- Podem existir vários blocos de partida, sendo estes distinguidos pela cor.
- Apenas se pode desdobrar o objeto caso este não vá sobrepor nenhum outro bloco ou saia do tabuleiro após o movimento.

Formulação do Problema

- Representação do estado:
  - Matriz x por y
  - Val = {-1,0,1,...}
    - -1 -> null space
    - 0 -> empty space
    - Val > 0 -> some color



- Estado inicial:
  - Matriz começa com pelo menos um Val = 0, e pelo menos um Val > 0.

1	0	-1
0	0	2
-1	-1	0

- Estado final:
  - Se não houver espaços a 0, jogador ganha
  - Se houver espaços a 0 e não for possível fazer mais operações, o jogador perde

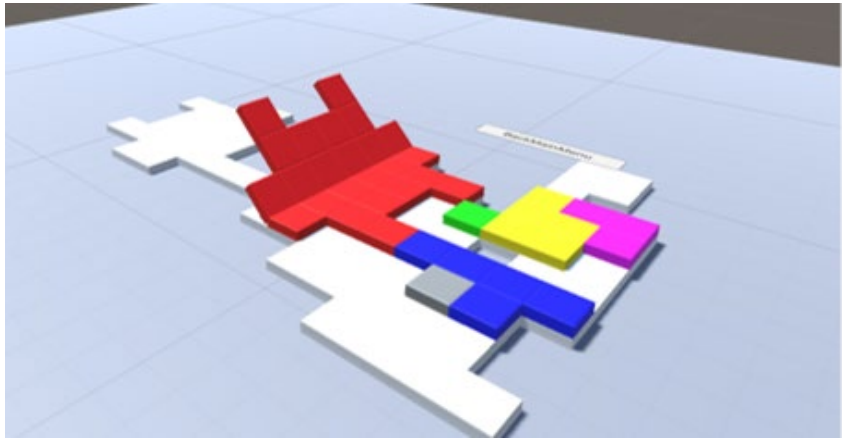
1	1	-1
1	1	2
-1	-1	2

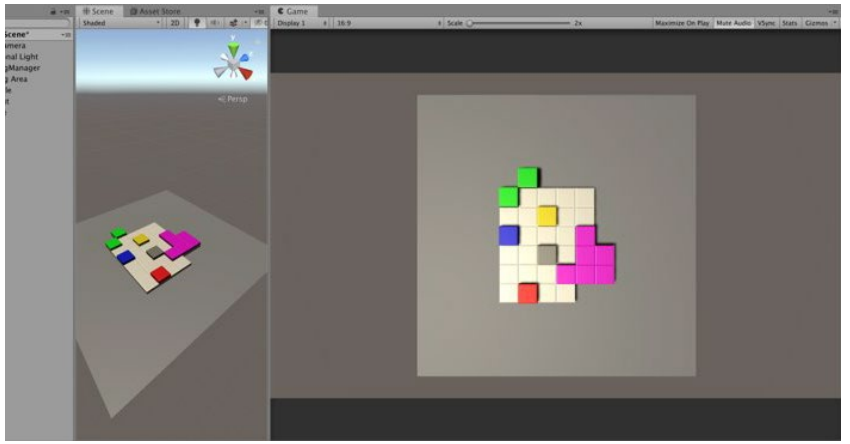
- Operadores:

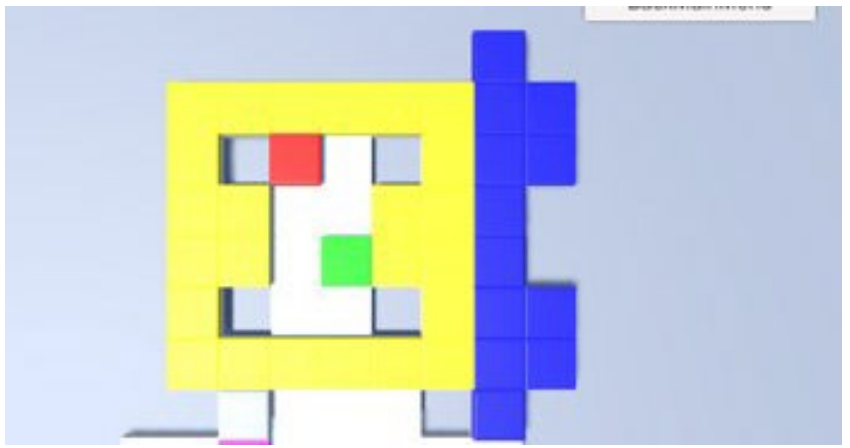
Nome	Pré-Condições	Efeitos
Flip Val up	Val > 0, axis = min(y) where Matrix[x][y] == Val, ∀piece ∈ Matrix[piece.x,2*axis-piece.y - 1] == 0	∀piece, Matrix[piece.x,2*axis-piece.y - 1] = Val
Flip Val down	Val > 0, axis = max(y) where Matrix[x][y] == Val, ∀piece ∈ Matrix[piece.x,2*axis-piece.y + 1] == 0	∀piece, Matrix[piece.x,2*axis-piece.y + 1] = Val
Flip Val right	Val > 0, axis = max(x) where Matrix[x][y] == Val, ∀piece ∈ Matrix[2*axis-piece.x - 1,piece.x] == 0	∀piece, Matrix[2*axis-piece.x - 1, piece.y] = Val
Flip Val left	Val > 0, axis = min(x), where Matrix[x][y] = Val, ∀piece ∈ Matrix[2*axis-piece.x + 1,piece.y] == 0	∀piece, Matrix[2*axis-piece.x + 1, piece.y] = Val

Trabalho desenvolvido

- Implementada uma interface gráfica aproximada da do jogo original.
- Implementado o modo de jogador Humano, com "dicas" caso o jogador esteja com dificuldade.
- Foi criado um gerador de puzzles, de modo a ser possível testar os resultados com puzzles sempre diferentes
- Foram implementados dois algoritmos de Aprendizagem por Reforço, PPO e SAC
- Comparação minuciosa dos vários algoritmos implementados, com diferentes funções de avaliação e com diferentes parâmetros

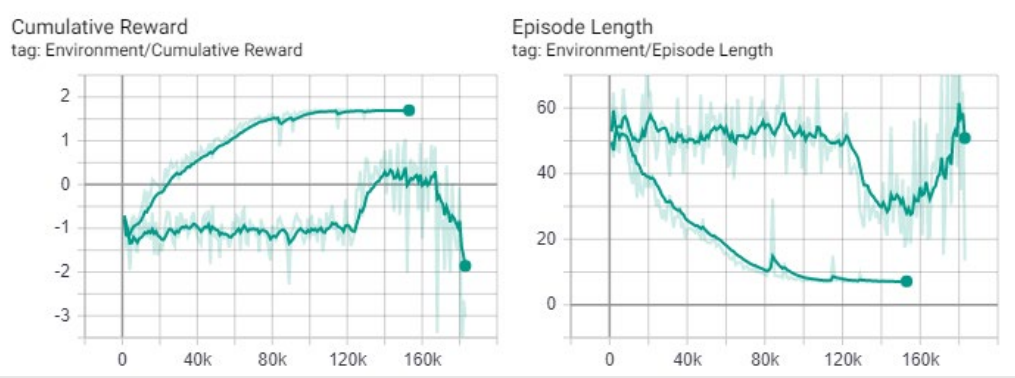






PPO vs SAC

- PPO tem uma taxa de puzzles bem sucedidos maior que a do SAC.
- Além disso convergiu consideravelmente mais rapidamente que este.
- Aprendeu rapidamente que moves eram válidos, em comparação com o SAC que parece não ter aprendido de todo.
- SAC necessita de mais moves para resolver o puzzle
- Ambos os Algoritmos não são muito genéricos, apenas resultando para os puzzles onde foram treinados.
- O PPO teve em geral uma melhor performance que o SAC



Training Set

Algoritmo	PPO	SAC
Taxa de Sucesso	100%	90%
Taxa Movimentos Válidos	97.12%	18%
Número médio movimentos per Puzzle	8.0	43.95
Tempo	2m 52s	3m 55s

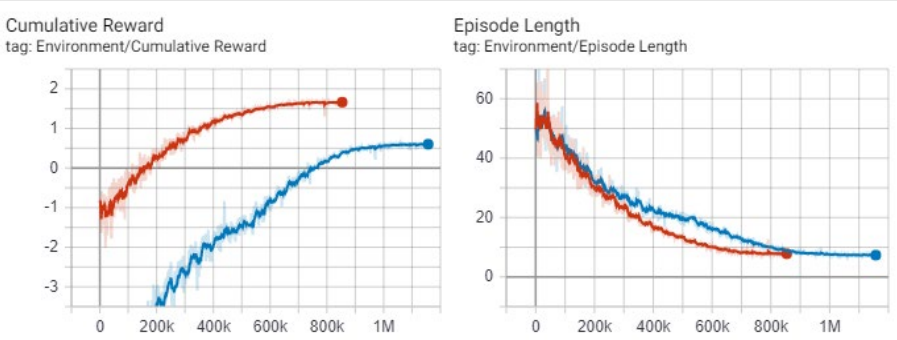
Test Set

Algoritmo	PPO	SAC
Taxa de Sucesso	0.5%	0%
Taxa Movimentos Válidos	1.8%	0.1%
Número médio movimentos per Puzzle	154.77	448.3

Funções de avaliação

Training Set	1	2
Taxa de Sucesso	100%	100%
Taxa Movimentos Válidos	98%	93%
Jogadas por Puzzle	8.5	8.8

Test Set	1	2
Taxa de Sucesso	0.5%	0%
Taxa Movimentos Válidos	1.4%	4.42%
Jogadas por Puzzle	213.45	71.84



Função de Avaliação	1	2
Puzzle Resolvido	1	1
Puzzle Falhado	-0.8	-0.8
Movimento Válido	0.1	-0.05
Movimento Inválido	-0.05	-0.15

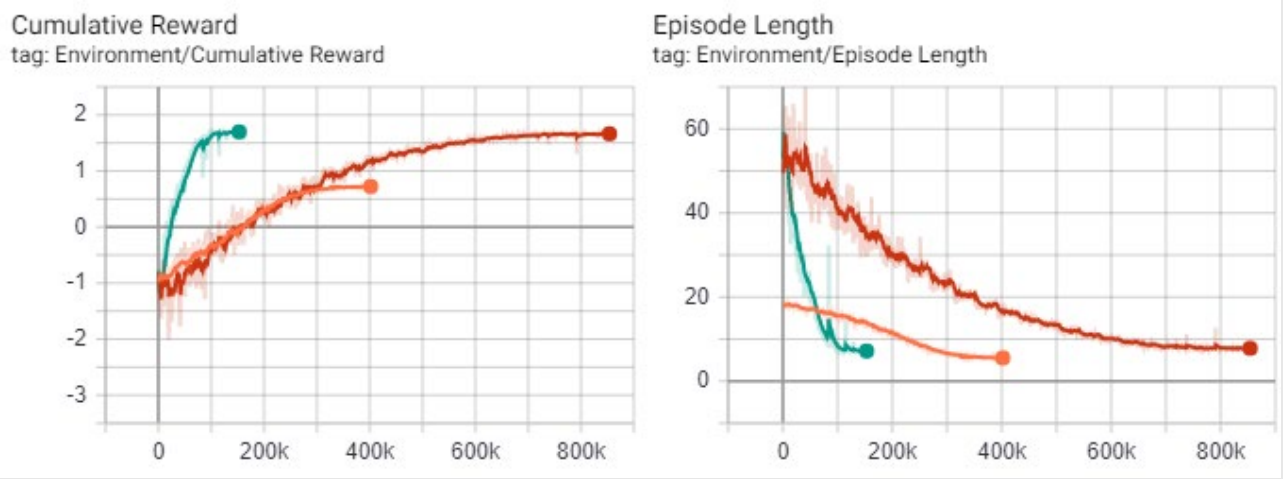
- Ambas as funções são bastante semelhantes.
- Na primeira função tentou-se dar uma pontuação negativa aos moves válidos de modo a tentar minimizar o número de moves
- Na segunda optou-se for uma valorização de moves válidos.
- Ambas atingiram excelentes resultado no training set.
- A Primeira conseguiu superar um pouco a segunda.
- Resultados geralmente maus no ambiente de teste
- A Primeira função acabou por obter mais moves válidos
- Número semelhante de jogadas por puzzle
- No ambiente de teste aconteceu precisamente o contrário
- A segunda função necessita de muitos menos moves em ambiente de teste.

Parâmetros PPO

Training Set	1	2	3
Taxa de Sucesso	100%	100%	100%
Taxa Movimentos Válidos	61.55%	98%	97.12%
Jogadas por Puzzle	13.99	8.5	8.0
Tempo	1h 17m	18m	3m

Test Set	1	2	3
Taxa de Sucesso	5%	0.5%	0.5%
Taxa Movimentos Válidos	2.1%	1.4%	1.8%
Jogadas por Puzzle	169.72	213.45	154.77

- Batch_size bastante mais pequeno do que o inicial, visto que é um espaço de ação discreto
- Buffer Size mais pequeno, de modo a aumentar a rapidez
- Inicialmente o número de steps não era suficientemente grande, sendo necessário por o valor máximo
- Sequence_length mais pequeno, visto que o agente não necessitava de se lembrar de coisas a muito longo prazo.
- O valor de Beta foi diminuído, de modo a tomar algumas ações aleatórias, mas não demasiadas
- Epslion a 0.3, de modo a acelarar o treino.
- 0.9 de modo a ele se concentrar na estimativa de valor mais atual

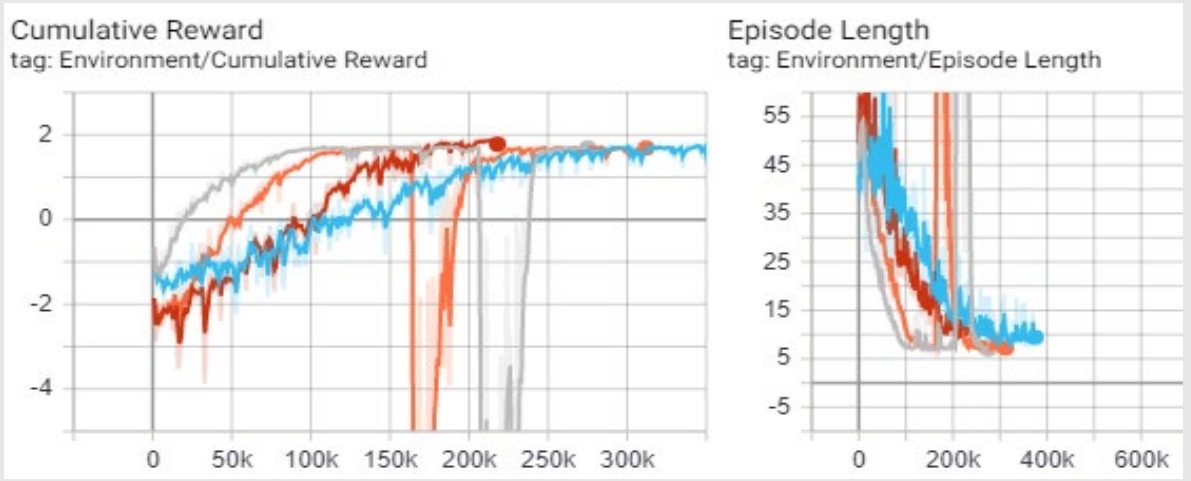


Comparação de dificuldades

Training Set	Fácil	Média	Difícil	Geral
Taxa de Sucesso	100%	100%	100%	99%
Taxa Movimentos Válidos	99%	99%	98%	63.16%
Jogadas por Puzzle	8.2	8.12	9.06	14.08
Tempo	5m 53s	4m 4s	6m 15s	7m

Test Set	Fácil	Média	Difícil	Geral
Taxa de Sucesso	0%	0%	0%	0.2%
Taxa Movimentos Válidos	4.1%	0.6%	1.8%	0.2%
Jogadas por Puzzle	73.16	434.19	563.5	654.13

- A diferença de dificuldade para um puzzle do mesmo tamanho não é muito significativa.
- Todos estes testes foram testados com puzzles de tamanhos superior aos testes anteriores
- Os puzzles de dificuldade mais baixa convergiram mais cedo.
- Mesmo após de terem convergido nota-se no gráfico uma forte componente de exploração.
- Isto pode ser explicado por alguns parâmetros que temos para o PPO.
- O Solver geral teve resultados ligeiramente piores, sendo que consegue resolver puzzles de todas as dificuldades.



Conclusões

- O PPO provou ser mais adequado do que o SAC no Folding Blocks.
- As duas melhores funções de avaliação, por não serem muito diferentes, acabaram por ter resultados bastante semelhantes.
- O Q-Learning e o SARSA provavelmente teriam ainda melhor resultados e seria interessante fazer esta abordagem.
- Os parâmetros dados ao PPO acabaram por influenciar de uma maneira bastante significativa o desempenho do algoritmo
- Apesar de o puzzle aprender rapidamente como resolver um puzzle onde já treinou, não foi possível generalizar para outros.
- Percebemos que é bastante mais difícil ter bons resultados com RL do que com Supervised Learning, sendo no entanto uma das áreas atuais com mais potencial.
- Pensamos que conseguimos atingir os objetivos propostos, tendo aprendido bastante sobre esta área.

Referências

Embora não tenhamos encontrado nenhum código fonte/artigo referente exatamente ao nosso jogo, encontramos alguns artigos em que são mencionadas heurísticas que achamos pertinentes aplicar na nossa resolução:

- [1] S. McAleer, F. Agostinelli, A. Shmakov, P. Baldi , Solving the Rubik'sCube Without Human Knowledge
- [2] D. Budakova, V. Vasilev , Applying Reinforcement learning to find thelogic puzzlessolutionTechnical University of Sofia
- [3] B.Bischoff1,D.Nguyen-Tuong and H.Markert1,A.Knoll //Solving the 15-Puzzle GameUsing Local Value-IterationJun 2012
- [4] Borga, M Reinforcement Learning Using Local Adaptive Models.ISY, Link ooping University, ISBN 91-7871-590-3, LiU-Tek-Lic-1995:39(1995)
- [5] Noppon Choosri, Solving Sheduling Problems as the Puzzle Games UsingConstraint ProgrammingOctober 2004
- [6] Sutton, R. S., Barto, A. G.: Reinforcement Learning: An Introduction(AdaptiveComputation and Machine Learning). The MIT Press (1998)
- [7] Y. Wang and H.He,C.Wen X. Tan .Truly Proximal Policy Optimization.In Proc. of AAAI-94, pp. 301–306.
- [8] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling.The arcade learningenvironment: An evaluation platform for general agents. Journal ofArtificial IntelligenceResearch, 47:253–279, 2013

Tecnologias Usadas

Na realização deste trabalho foi usado o Ml_Agents toolkit,
Unity para a Interface Gráfica e C# para a lógica do jogo