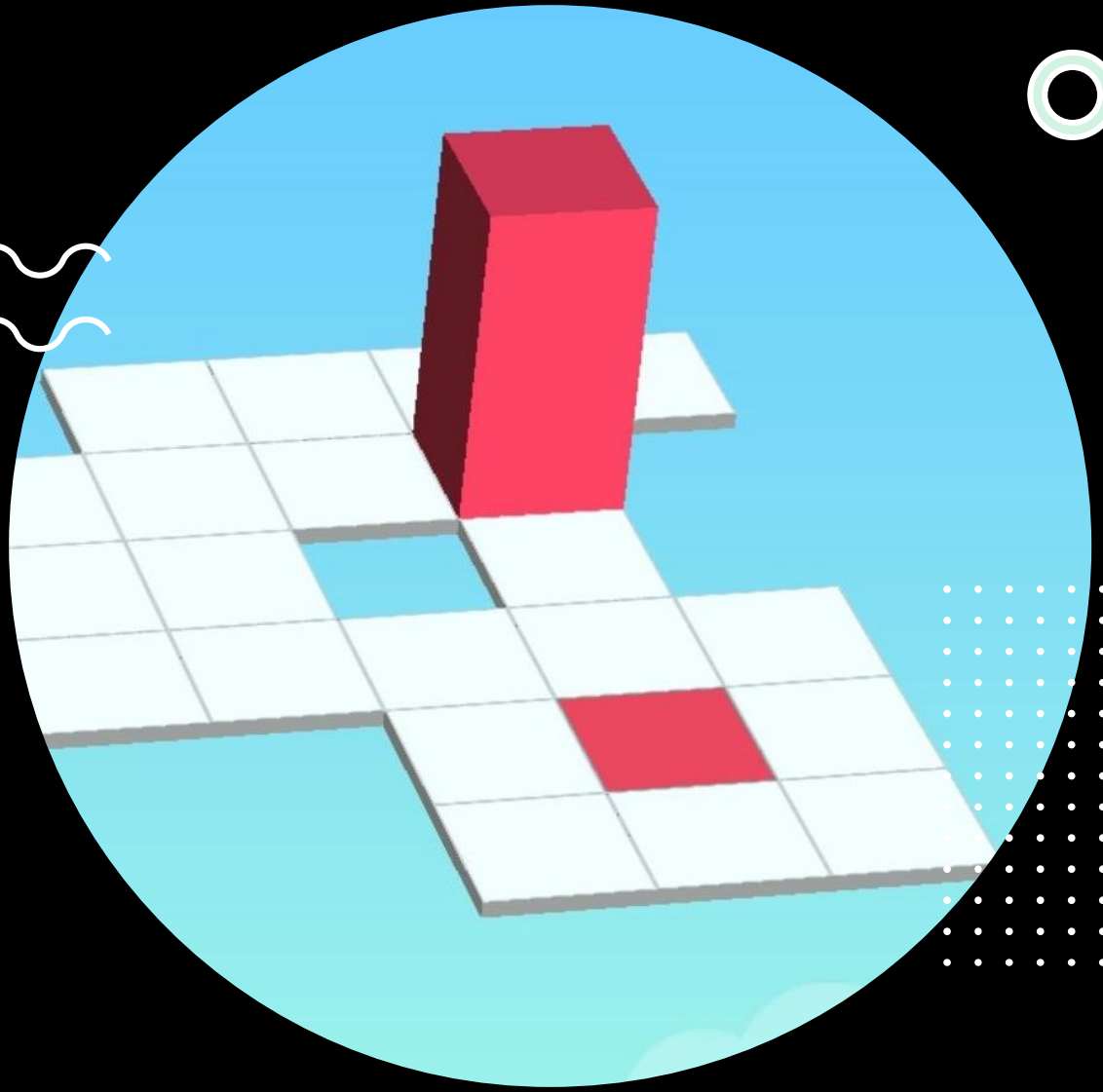
BLOXORZ
ROLL THE BLOCK

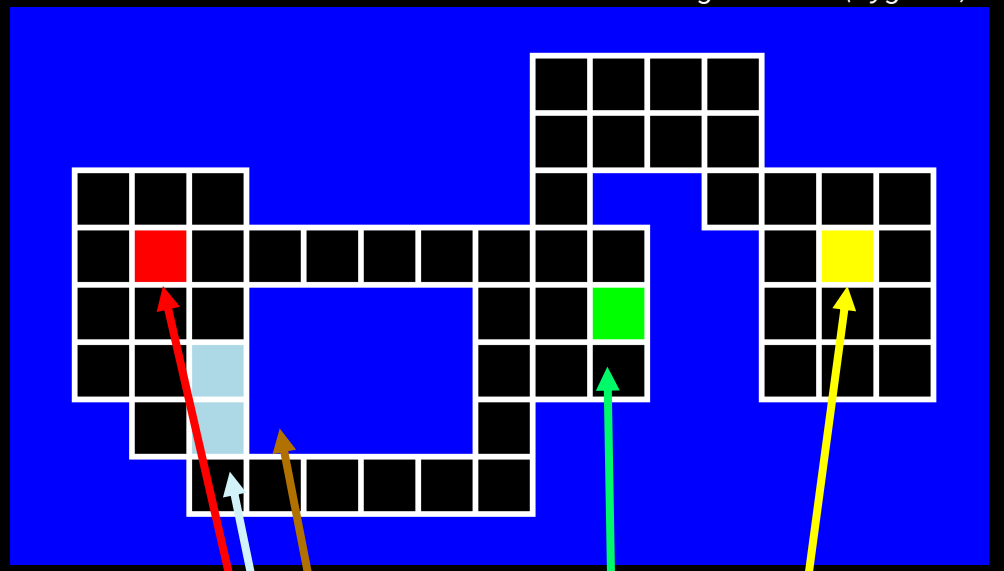


FRANCISCO TAVARES
RODRIGO BATISTA

Introdução

- O jogo consiste num bloco em forma de retângulo, que tem como objetivo chegar a um ponto definido no tabuleiro no menor numero possível de movimentos, ou seja, com a solução ideal
- No percurso também há, em alguns níveis, quadrados especiais como botões que desbloqueiam quadrados para o percurso e quadrados de “vidro” em que se os ocuparmos no estado *vertical*, estes partem e o bloco cai no vazio

Fig 1- Nível 9 (Pygame)



```
[[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0],
 [0,1,1,1,1,0,0,0,0,0,1,0,0,1,1,1,1,1,0],
 [0,1,2,1,1,1,1,1,1,1,1,0,0,1,9,1,1,1,0],
 [0,1,1,1,1,0,0,0,0,1,1,5,0,0,1,1,1,1,0],
 [0,1,1,4,1,0,0,0,0,1,1,1,0,0,1,1,1,1,0],
 [0,0,1,4,51,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
 [0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0],
 [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]
```

Fig 2- Nível 9 (Matriz)

Legenda:

0 – vácuo
1 – caminho
2 – retângulo em pé
3 – retângulo deitado
4 – chão de vidro
5 – botão pesado
6 – botão ativado
9 - meta

O jogo como um problema de pesquisa

- O bloco começa sempre na vertical
- O estado objetivo é chegar ao quadrado definido e acabar nele na vertical
- O bloco pode estar em três estados diferentes, o *vertical* quando está em pé, o *horizontal1* quando está deitado na vertical e o *horizontal2* quando está deitado na horizontal
- As precondições para todos os movimentos é a necessidade de os quadrados para onde o bloco se vai mover terem de existir, até no caso do movimento *horizontal* em que é necessário existirem ambos os blocos e não só um
- O custo é sempre 1 que é o movimento do bloco

Fig.3-Vertical

Fig.4-Horizontal1

Fig.5-Horizontal2

Níveis, dificuldades e jogabilidade

- À medida que os niveis avançam, a dificuldade e a complexidade dos níveis aumenta.
- Distribuimos os níveis em duas partes: níveis cujo computador consegue resolver e níveis cujo computador não consegue resolver (Estes níveis possuem adições que o computador não deteta).

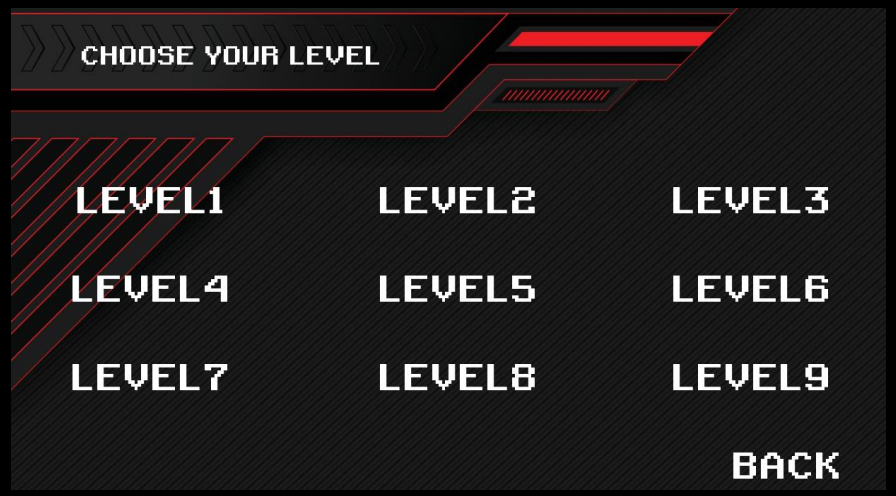


Fig 6- Menu dos níveis (Pygame)

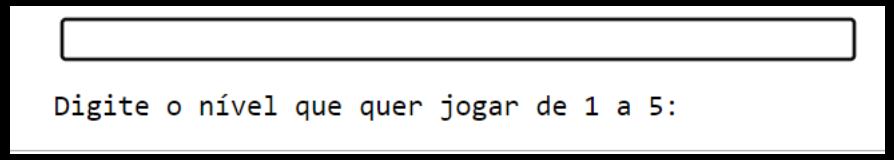


Fig 7- Escolha dos níveis (terminal)

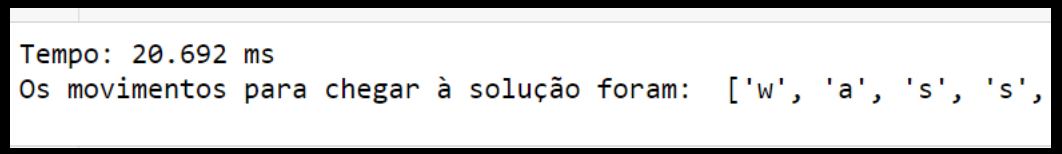


Fig 8- Resultado de um algoritmo de pesquisa (terminal)

Sobre o Código implementado

- Na implementação do código decidimos não usar classes e sim funções, dividindo assim o jogo em várias funções pequenas para ser de mais fácil leitura e interpretação.

- Conseguimos implementar todos os níveis em Pygame. No entanto, relativamente à parte de pesquisa não é possível ver o computador a resolver no Pygame (apenas no terminal).

- Diversas funções foram baseadas nas funções do moodle, fornecidas pelo professor (exemplo desta foi o desenho do nível em Pygame).

- Usamos ainda algumas bibliotecas fundamentais, como a biblioteca copy(deepcopy) e pygame (mais aprofundado no ReadMefile)

Algoritmo e heurística

- Neste trabalho decidimos implementar 4 algoritmos de pesquisa sendo eles o BFS(Breadth First Search), o DFS(Depth First Search), o Greedy search e o A* search.

- Relativamente ao Greedy search e ao A* search usamos uma heurística, a distância de Manhattan, que é calculada através da soma das distâncias das coordenadas x e y entre a posição atual do bloco e a meta. Como no nosso jogo temos um estado em que o bloco ocupa duas “grids”, nesse caso, a distância é o mínimo das distâncias desses blocos até à meta.

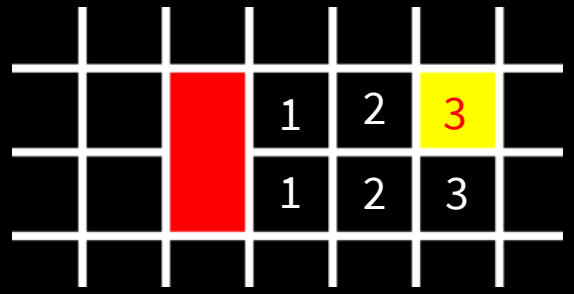
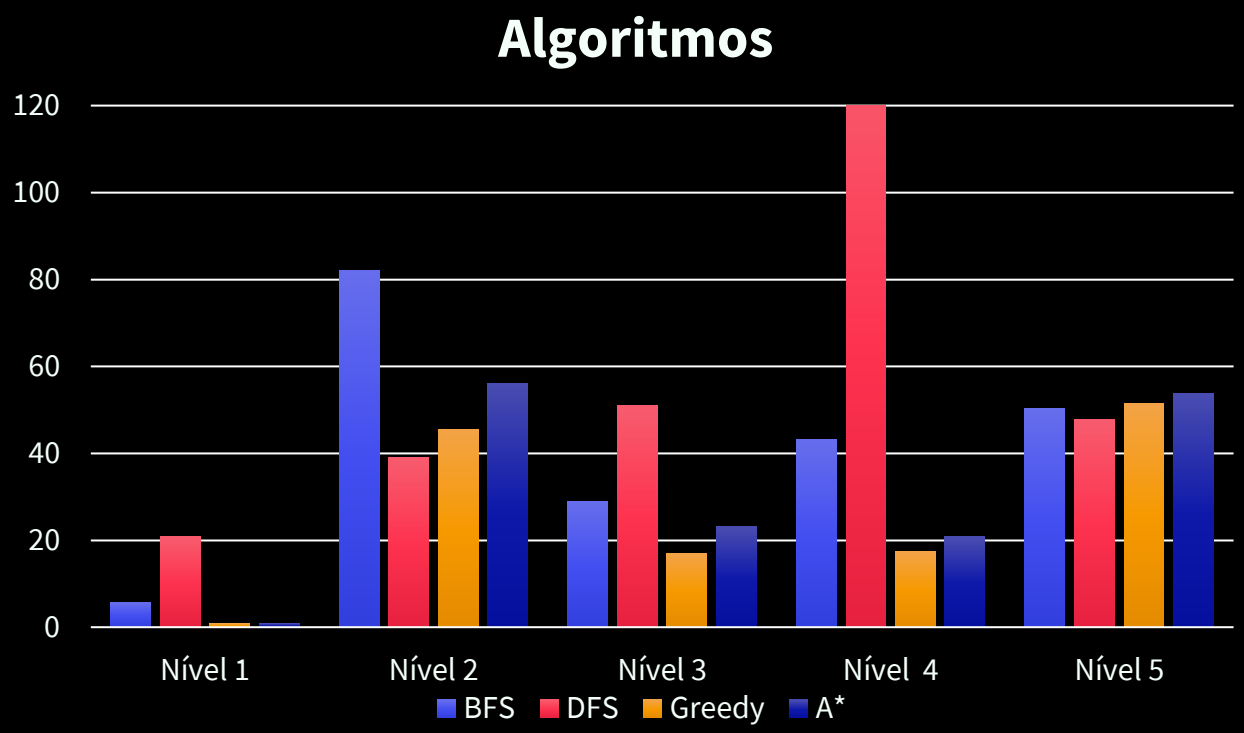


Fig 9- Representação do bloco e da meta

Comparação e resultados experimentais



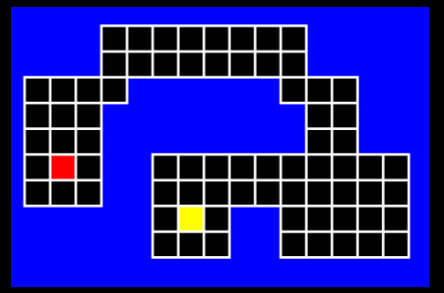


Fig 10- Nível 2

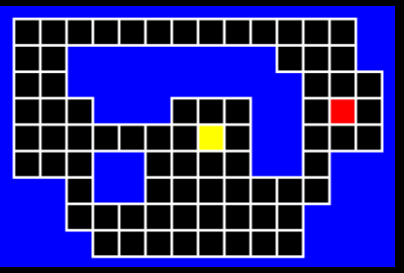


Fig 11- Nível 4

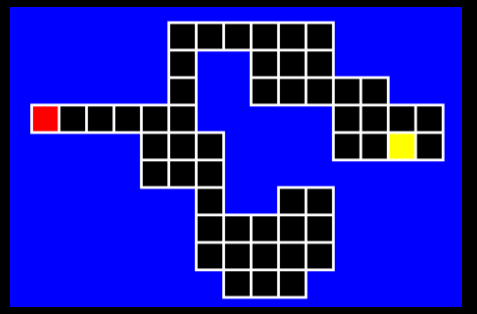


Fig 12- Nível 5

Discussão dos resultados experimentais

- Para a obtenção destes resultados experimentais foram feitos 5 testes para cada algoritmo de pesquisa, em cada nível, sendo de seguida calculada em média.
- Foi depois feita esta tabela (e o gráfico visto no slide anterior) onde as células a verde representam a solução ótima de cada nível.
- Através dos resultados obtidos podemos observar que o algoritmo de pesquisa menos eficiente para este jogo é o DFS, pois ele visita bastantes nós antes de chegar à solução ótima. Por sua vez o Greedy search, não é muito eficiente neste jogo, pois apenas encontrou a solução ótima uma vez. Já o BFS apesar de demorar um pouco mais que o A*, como os níveis não eram muito complexos, encontrou bastante rápido a solução, sendo assim uma opção viável para este jogo. Por último o A*, que, não sendo tão rápido como o greedy, encontra sempre a solução ótima

	BFS	DFS	Greedy	A*
Nível 1	5,788	20,895	0,997	0,995
Nível 2	82,135	39,132	45,508	56,172
Nível 3	28,983	51,001	16,882	23,164
Nível 4	43,324	131,747	17,377	20,865
Nível 5	50,341	47,916	51,421	53,821

Tabela 1- Resultados experimentais

Conclusão

- Em conclusão, este trabalho explorou o jogo RollTheBlock como um problema de pesquisa e aplicou diferentes algoritmos de busca para encontrar soluções eficientes para o jogo.

- Neste estudo, o BFS, o DFS, o greedy e o A* foram testados para encontrarem soluções. Os resultados indicaram que o algoritmo A* obteve o melhor desempenho em termos de qualidade das soluções encontradas, enquanto a busca em profundidade teve o pior desempenho.

- Além disso, este estudo também mostrou que a distância de Manhattan (para este jogo) melhora o desempenho do algoritmo A* e mostrou-se eficaz para encontrar soluções mais rapidamente.

- Por fim, este estudo demonstrou que o uso de algoritmos de pesquisa pode ser uma ferramenta valiosa para a resolução de problemas de jogos, fornecendo soluções eficientes e otimizadas. No entanto, é importante considerar as limitações de cada algoritmo e escolher aquele que melhor se adapta às características específicas do problema em questão.

Bibliografia

- https://www.youtube.com/watch?v=BT2cjrxGpWo&list=PLJ8PYFcmwFOxtJS4EZTGEPxMEo4YdbxdQ
- https://www.youtube.com/watch?v=GMBqjxcKogA
- https://www.youtube.com/watch?v=lGjV2dvuDKs
- https://chat.openai.com/chat
- Ficheiros das aulas práticas