TPG-RUSH HOUR

Miguel Gomes - 103826
Diogo Pires - 97889

Introdução

Resolução do jogo "Rush Hour" usando inteligência artificial. Todo o código foi desenvolvido no ficheiro **student.py e search.py**

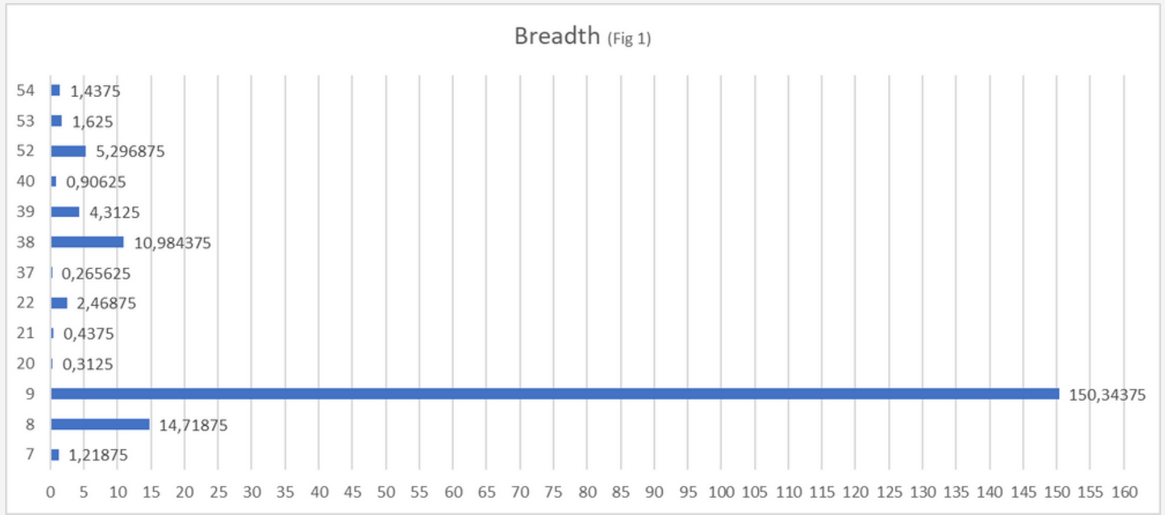
Começámos por testar o funcionamento do jogo, anotando todos os detalhes úteis para a resolução do problema. O carro principal (*player car*) é sempre representado por 'AA', os restantes são representados por
 outras letras maiúsculas do alfabeto, os blocos de parede são representados por 'x' e os blocos livres são
representados por 'o'.

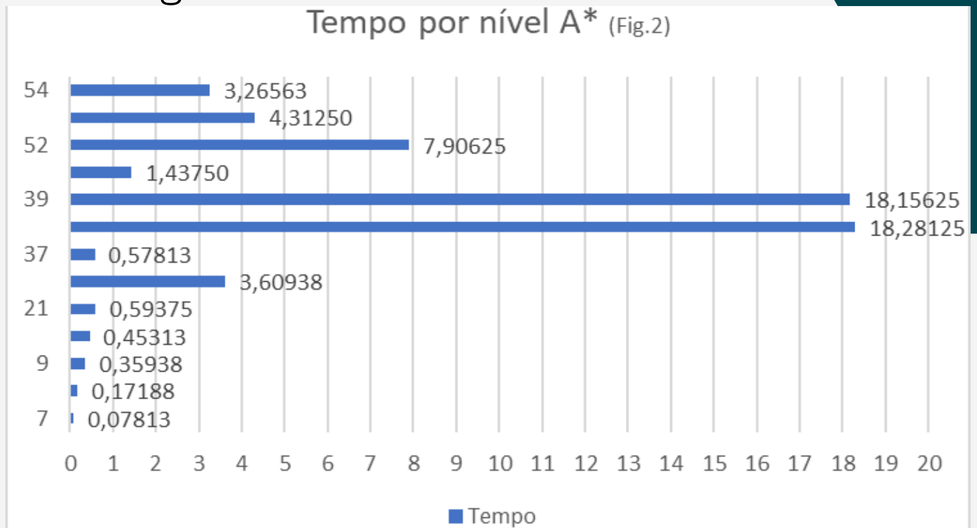
Algoritmo

- Com base no que aprendemos nas aulas práticas elaborámos uma árvore de pesquisa para nos ajudar na solução deste problema. Nesta árvore elaborámos funções para obter o caminho da árvore de pesquisa e obter a lista de ações possíveis num determinado estado.
- Depois implementámos a função de pesquisa propriamente dita onde adicionámos a uma lista os melhores estados garantindo que não havia repetições de estados.
- Para a segunda parte do projeto, melhorámos o nosso algoritmo e introduzimos a heurística para introduzir a pesquisa em A* sempre que tal se justificasse. Usámos pesquisa em largura para os níveis mais pequenos porque é um método mais célere que a pesquisa A*, esta última foi usada para resolver níveis de maior dimensão.

Heurística

- A heurística utilizada foi com base nos veículos que bloqueiam a saída. Consideramos primeiro o carro que bloqueia o "*player car*", depois, os veículos que impedem esse mesmo carro de se mover. A heurística utilizada é então a soma dos carros que bloqueiam o "*player car*" com os carros que bloqueiam o carro "bloqueante".
- Em baixo temos um gráfico utilizando pesquisa do tipo "*breadth*" (FIg.1) e um gráfico utilizando pesquisa do tipo A* utilizando a heurística descrita acima (Fig2). Com estes gráficos podemos comparar a diferença entre os dois tipos de pesquisas principalmente nos níveis de maior dimensão.
- Por fim concluímos que efetivamente a pesquisa do tipo "*breadth*" é mais eficiente que a pesquisa do tipo A* em níveis de menor dimensão, daí termos optado por apenas usar A* em níveis maiores de modo a otimizar o nosso algoritmo.





Melhorias

- Algumas melhorias que possamos implementar devem passar por, por exemplo, tentar reduzir o número de ciclos no nosso código de modo a tornar o código mais célere, melhorar alguns aspetos menos bem conseguidos da árvore de pesquisa e/ou tornar a heurística mais eficiente.