Inteligência Artificial

Algoritmos de procura aplicado ao VectorRace

João Almeida - A95191
Daniel Du - A97763
Afonso Amorim - A97569


João Almeida


Daniel Du


Afonso Amorim

Grupo 50
30 de novembro de 2022

Conteúdo

1 Introdução **1**

2 Descrição do problema **1**
2.1 Legenda de um circuito 1
2.2 Regras 1

3 Formulação do problema **1**
3.1 Carro 1
3.2 Estado inicial 2
3.3 Estado final 2
3.4 Operações 2
3.4.1 Calcular a Próxima posição 2
3.4.2 Próximo estado 2
3.4.3 Algoritmos usado para verificação de colisão de paredes 2
3.5 Custo da solução 3

4 Representação da pista em forma de grafo **3**

5 Pistas e análise de resultados **3**

6 Algoritimo escolhido e estratégia aplicada **5**

7 Conclusão **5**

1 Introdução

Este relatório surge na resolução do exercício em grupo da unidade curricular de Inteligência Artificial.

O exercício proposto basea-se na elaboração de algoritmos de procura para a resolução de um jogo, nomeadamente o *VectorRace*, também conhecido como *RaceTrack*.

Nesta primeira fase do projeto, iremos desenvolver um circuito com um participante a encontrar um caminho, preferencialmente, o mais curto através dos algoritmos de procura em largura e em profundidade.

Para a resolução deste exercício, usaremos a linguagem *Python*.

2 Descrição do problema

Nesta fase do projeto foi-nos pedido tarefas simples, como por exemplo, criar, pelo menos, um circuito VectorRace, a representação de uma pista em forma de grafo e desenvolver uma estratégia de procura. A pista é um conjunto de linhas e colunas, onde temos diversas posições (linha,coluna), o que facilita bastante o uso de vetores.

2.1 Legenda de um circuito

- X representam as paredes
- P representam a posição inicial
- F representam a posição de chegada
- \- representam caminho possível

2.2 Regras

- O carro pode acelerar -1, 0 ou 1 unidades em cada direção (linha,coluna).
- Se o carro sair da pista, o carro terá de voltar à posição anterior, assumindo um valor de velocidade igual a 0, aumentando em 25 o custo.

3 Formulação do problema

Dada a descrição acima, decidimos dividir o problema por parâmetros de modo a simplicar a formulação do mesmo.

Temos então, de forma sucinta:

3.1 Carro

Criamos uma classe carro, em que cada carro tem como paramentros 2 tuplos, um é a velocidade e o outro é a posição do carro. O carro é futuramente passado para representar um nodo no grafo.

3.2 Estado inicial

O estado inicial é representado por um carro na posição inicial do mapa, isto é, onde está o carater *P*, e com uma velocidade (0,0).

3.3 Estado final

Para representar um estado final utilizamos uma lista de coordenadas onde se encontra o carater 'F' no mapa. Conseguimos saber se um carro se encontra no estado final quando as suas coordenadas tiverem na lista.

3.4 Operações

3.4.1 Calcular a Próxima posição

Operação onde o programa calcula a próxima posição do carro, dada uma aceleração e tendo em conta as fórmulas apresentadas no enunciado (velocidade e posição).

3.4.2 Próximo estado

Operação que avalia todas as hipóteses possíveis para a próxima posição. A função nextestados devolve uma lista carros nas posições possíveis de ir. Depois de calcular as posiçoes e velocidades possíveis com todos os vetores acelarações existentes filtramo-os:

- pelos que acabam numa parede
- pelos que passam por cima de paredes

3.4.3 Algoritmos usado para verificação de colisão de paredes

Para verificar a colisão do carro com paredes meio do caminho, calculamos a interseção de dois caminhos, cuja decomposição vetotiral da diferença das posições fazem 2 triangulos como se ve na figura, com paredes:

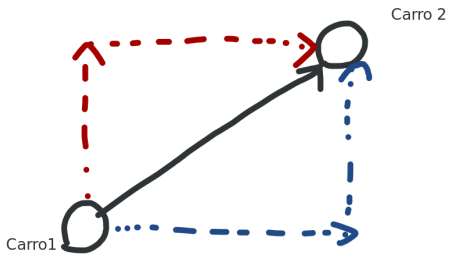


Figura 1: Pista 1

Na figura acima demonstra graficamente o algoritmo usado. Sendo que o carro 1 representa a posição inicial de um carro e o carro 2 é a posição final desse mesmo carro depois de mover. Calculamos a interseção da fronteira do triangulo azul com paredes e o triangulo bordo com paredes. Se ambas intersetarem paredes admitimos que o carro não pode deslocar-se. Basta haver um dos triangulos a não intersetar paredes para considerar a deslocação válido.

3.5 Custo da solução

O custo é calculado na função que gera os próximos estados possíveis. Cada movimento de um carro acrescenta 1 unidade ao custo e, caso ele saia da pista, são acrescentadas 25 unidades.

4 Representação da pista em forma de grafo

Fizemos vários circuitos e guardamos em ficheiros *.txt*. Fizemos uma função que transforma um circuito numa lista de listas, onde cada lista da lista "principal"contem carateres que formam uma, formando assim uma matriz de carateres.

Após isto, construímos um grafo que associa um nó a uma lista de outros nós, neste caso carros. Para fazermos este dicionário de listas de adjacência partimos do estado inicial e calculamos uma lista com os possíveis estados seguintes. Agora, no grafo o nó inicial é a chave para os valores dessa mesma lista e continuamos o processo para esses estados possíveis, até não haver mais nodos a visitar.

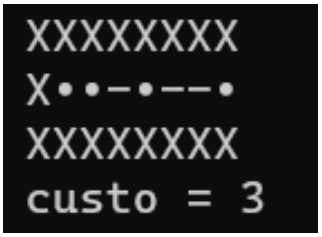
5 Pistas e análise de resultados

Nas imagens em baixo estão as pistas que usamos para testar os nossos algoritmos e as pistas após aplicar a função BFS, imagemdo lado esquerdo, e a função e a DFS, imagem do lado direito, onde o carater '•' representa as posições por onde o carro

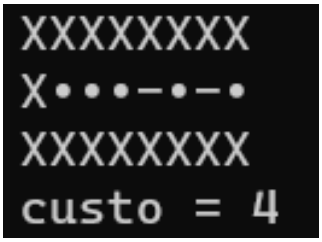
A pista 1, representada na figura 2, como podemos ver é uma pista muito básica, é só uma linha reta.

Para este circuito, o tempo de geração do grafo e a execução do algoritmo é quase instantâneo.

Figura 2: Pista 1



BFS aplicada à pista 1



DFS aplicada à pista 1

De seguida, a pista 2, representada na figura 3, embora ainda sem paredes ou obstáculos no meio do circuito, já implica o carro andar em diferente direção e sentido.

Para este circuito tempo de geração do grafo e correr o algoritmo é aproximadamente 2 segundos.

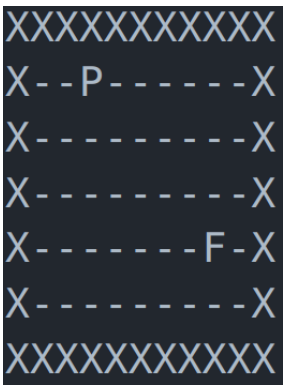


Figura 3: Pista 2

BFS aplicada à pista 2

DFS aplicada à pista 2

Por último, temos a pista 3, a mais complexa utilizada até ao momento, já implica mudar de direção e sentido e com paredes no meio do circuito.

Para este circuito tempo de geração do grafo e correr o algoritmo é aproximadamente 2 min e 40 segundos.

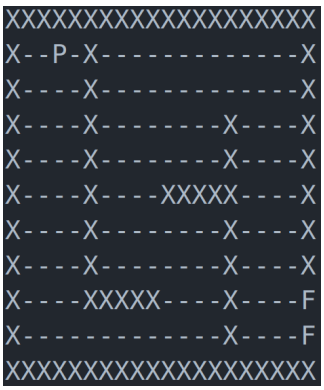
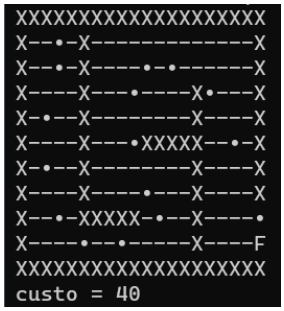
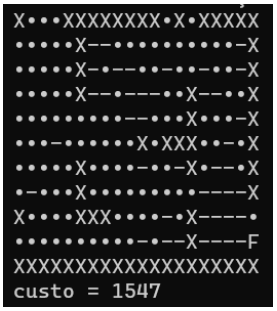


Figura 4: Pista 3



BFS aplicada à pista 3



DFS aplicada à pista 3

Do mesmo modo que definimos desenvolver o trabalho de forma gradual, decidimos aumentar a dificuldade das pistas gradualmente também.

6 Algoritimo escolhido e estratégia aplicada

Para o nosso trabalho decidimos implementar algoritmos de procura não informada, mais concretamente, os algoritmos *BFS (Breath First Search)* e *DFS (Depth First Search)*.

O *BFS* ocupa muita memória e é muito demorado para problemas de grandes dimenções. Neste contexto o algoritmo percorre em todas as direções ao mesmo tempo cada nível de cada vez ate chegar ao destino e apresenta a solução com menos etapas para chegar ao destino

O *DFS* ocupa pouca memória para problemas com muitas soluções, a desvantagem é que a sua solução não é ótima, apresenta a primeira solução que encontrar seja esta boa ou má. Neste problema o algoritmo escolhe sempre um caminho aleatorio a percorrer até chegar ao destino.

7 Conclusão

Em suma, mesmo tendo em conta as adversidades que fomos encontrando ao longo do desenvolvimento do projeto, consideramos que fomos capazes de contornar os obstáculos que foram aparecendo.

Em relação ao projeto em si, sentimos que ficou aquém da nossa expectativa, visto que que o nosso grupo é capaz de mais e melhor e os resultados obtidos não corresponderam ao plano inicialmente tracejado pelo grupo.

No que toca aos algoritmos, de momento, apesar de retornarem um caminho aceitável, são pouco eficazes, isto é, demoram mais tempo do que expectado, tornando-se assim um ponto fulcral a melhorar para próxima fase.