Universidade do Minho
Escola de Engenharia

Inteligência Artificial
2022/2023

GRUPO 49

Artur Carneiro Neto de Nóbrega Luís A95414
Francisco Pinto Lameirão A97504
Lara Beatriz Pinto Ferreira A95454

A95414

A95454

A95454

2ª FASE
Dezembro de 2022

Índice

1 Introdução

Neste trabalho, proposto no âmbito da unidade curricular de *Inteligência Artificial*, desenvolveremos diversos algoritmos de procura para a resolução do jogo VectorRace, também conhecido como RaceTrack, de modo a conseguirmos resolver problemas através da conceção e implementação de algoritmos de procura.

Numa primeira fase, geramos um circuito VectorRace, com a indicação dos limites da pista, da posição inicial, que será o **I**, e da linha da meta, que corresponde ao **F**, em que o jogador teria de chegar ao **F** a partir de **I** da forma mais eficiente, seguindo assim, o caminho mais curto.

Para tal propósito, aplicamos os algoritmos de procura (não informada) BFS e DFS.

Na segunda fase, é pretendido desenvolver circuitos VectorRace com maior complexidade, implementando diversos algoritmos de procura informada e não informada (Deep-First Search, Breath-First Search, A estrela (A*), Gulosa). Além disso, nesta fase é pedido apresentar o ambiente com, pelo menos, dois participantes.

2 Descrição do Problema

Como referido anteriormente, o VectorRace, é um jogo de simulação de carros simplificado, que contém um conjunto de movimentos e regras associadas, de modo a um carro conseguir completar o circuito (deslocar-se do ponto I a F) percorrendo o caminho mais curto possível, evitando também bater nas "paredes"que contornam a pista.

O movimento do carro no VectorRace é bastante simples, tendo como base que as ações desenvolvidas são equivalentes a um conjunto de acelerações. Considerando a notação l, que representa a linha e c a coluna para os vetores, num determinado instante, o carro pode acelerar -1 ,0 ou 1 unidades em cada direção (linha e coluna). Consequentemente, para cada uma das direções o conjunto de acelerações possíveis é Acel = -1, 0, +1, com a = (al,ac) a representar a aceleração de um carro nas duas direções num determinado instante.

Tendo em conta que p como tuplo que indica a posição de um carro numa determinada jogada j (pj = (pl, pc)), e v o tuplo que indica a velocidade do carro nessa jogada (vj = ((vl,vc)) , na seguinte jogada o carro irá estar na posição:

$$pl^{(j+1)} = pl^j + vl^j + al$$
$$pc^{(j+1)} = pc^j + vc^j + ac$$

A velocidade do carro num determinado instante é calculada:

$$vl^{(j+1)} = vl^j + al$$
$$vc^{(j+1)} = vc^j + ac$$

Sendo uma simulação de corrida de carros, existe a possibilidade de o carro sair da pista, sendo que quando essa situação ocorrer o carro terá de voltar para a posição anterior, assumindo um valor de velocidade zero.

Cada movimento de um carro numa determinada jogada, de uma determinada posição para outra, terá um custo de 1 unidade, sendo que quando o mesmo sair dos limites da pista, o custo é de 25 unidades.

3 Formulação do Problema

3.1 Estado Inicial

I representa a posição inicial do jogador, de onde o carro deve partir e iniciar a sua trajetória até à meta.

A velocidade é nula.

3.2 Estado Final

F representa a posição final do jogador, a meta. O objetivo do jogador é chegar a F.

A velocidade, nesta fase final, não é relevante para a vitória do jogador.

3.3 Operadores

Nome: Subir

Pré-condição: O espaço que o jogador pretende ocupar não ser parede ou obstáculo.

Efeito: O carro sobe uma linha.

Custo: 1

Nome: Descer

Pré-condição: O espaço que o jogador pretende ocupar não ser parede ou obstáculo.

Efeito: O carro desce uma linha.

Custo: 1

Nome: Deslocar para a esquerda

Pré-condição: O espaço que o jogador pretende ocupar não ser parede ou obstáculo.

Efeito: O carro passa para a coluna à esquerda da posição anterior.

Custo: 1

Nome: Deslocar para a direita

Pré-condição: O espaço que o jogador pretende ocupar não ser parede ou obstáculo.

Efeito: O carro passa para a coluna à direita da posição anterior.

Custo: 1

Nome: Colidir com parede ou obstáculo

Pré-condição: O espaço que o jogador pretende ocupar ser parede ou obstáculo.

Efeito: O carro volta para a posição anterior à colisão.

Custo: 25

3.4 Custo Total da Solução

O custo total da solução, será equivalente à soma de todos os movimentos que o carro faz para chegar à meta. Cada movimento tem o custo de 1 unidade. Se bater contra uma parede, o custo é de 25 unidades.

4 Menu

Nesta segunda fase, decidimos mudar o menu para algo mais complexo e apelativo. O menu passa a apresentar 7 opções diferentes:

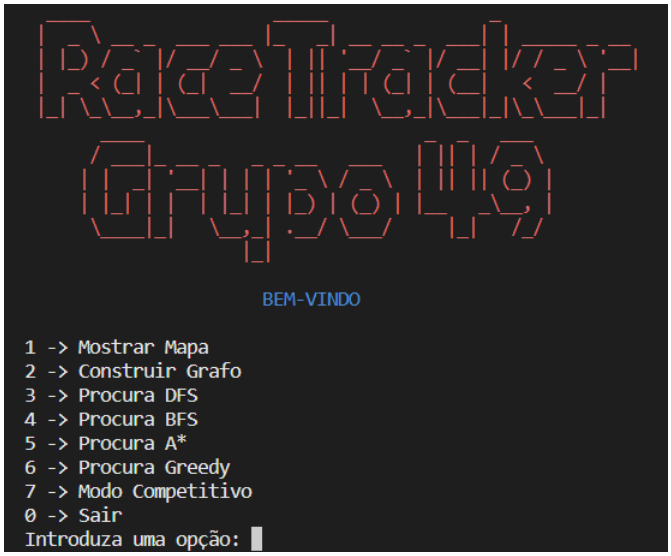

Figura 1: Menu

- **1) Mostrar Mapa** : Primeiramente, pede as dimensões do circuito e, em seguida, se os valores forem aceites, imprime no terminal um circuito aleatório com as dimensões fornecidas.
- **2) Construir Grafo** : Constrói o Grafo correspondente ao Mapa que é introduzido anteriormente.
- **3) Procurar DFS**: Imprime no terminal a solução correspondente à procura DFS.
- **4) Procurar BFS**: Imprime no terminal a solução correspondente à procura BFS.
- **5) Procurar A***: Imprime no terminal a solução correspondente à procura A*.
- **6) Procurar Greedy**: Imprime no terminal a solução correspondente à procura Greedy.
- **7) Modo Competitivo**: Entrar no Modo Competitivo com 2 participantes.
- **0) Sair**: Saímos do Menu.

Para as opções de imprimir qualquer tipo de solução no terminal (opções **3**, **4**, **5** e **6**) é imperativo construir primeiro o mapa e depois grafo (opção **1** e **2**).

5 Geração do Circuito

Nesta segunda fase da realização do trabalho, implementamos um algoritmo que gera os circuitos de forma aleatória, após o utilizador inserir as dimensões do mesmo, ou seja, largura e altura.

Em, seguida, o utilizador deve selecionar a opção de "Construir Grafo"(opção 2 do menu), para poder resolver o problema com sucesso.

Apresentamos, de seguida, alguns exemplos de mapas aleatórios que podem ser gerados para o jogo.

Figura 2: Circuito aleatório de dimensão 10x10

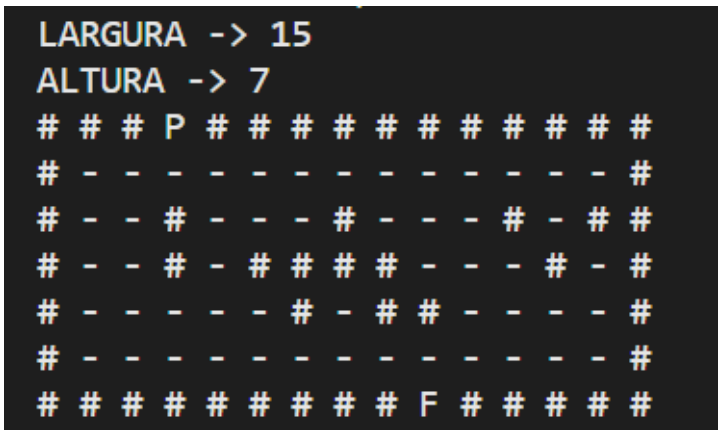

Figura 3: Circuito aleatório de dimensão 15x7

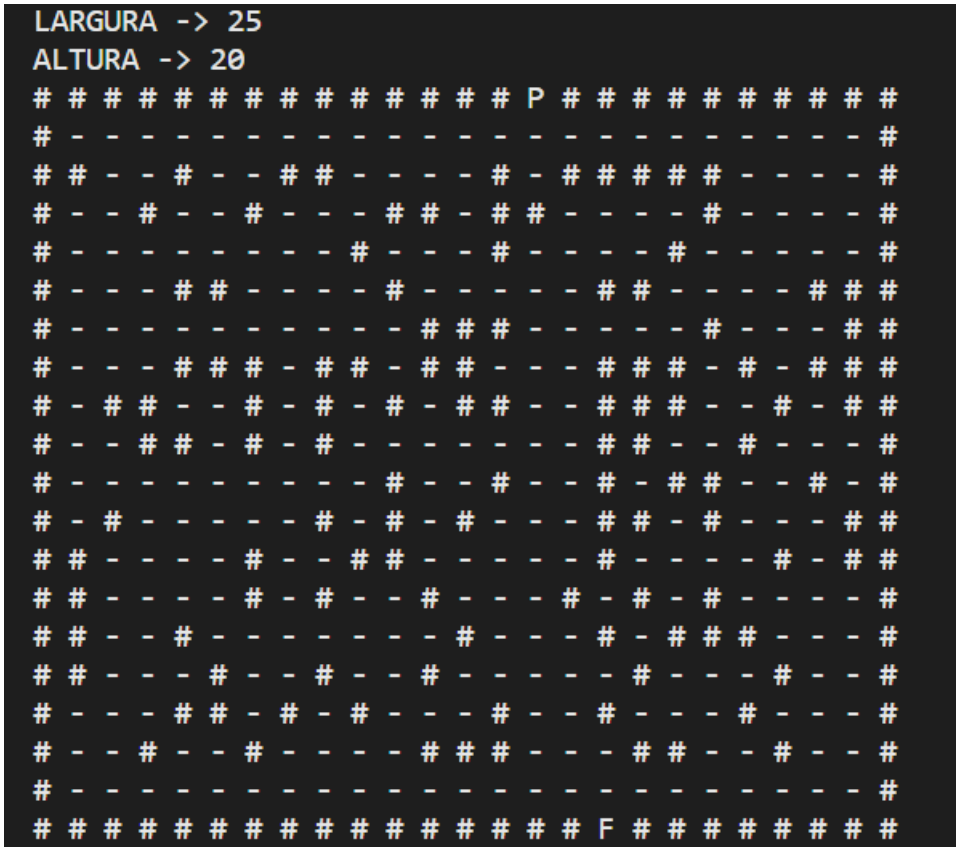

Figura 4: Circuito aleatório de dimensão 25x20

6 Grafo

Na construção do grafo, utilizamos duas funções: *constroiGrafo* e *expande*.

A função *constroiGrafo* é utilizada para criar a ligação no grafo entre o nodo atual e os nodos adjacentes que se encontram na lista que obtivemos na função *expande*, e para isso utiliza a função *adicionaAresta* que se encontra no *Grafo.py*.

A função *expande* é utilizada para expandir o estado atual, isto é, a função recebe o nodo atual e devolve uma lista com todos os nodos adjacentes que sejam viáveis para mover o carro na próxima jogada. Se o nodo adjacente for uma parede do circuito ou então estiver fora do mesmo, este não é adicionado à lista.

7 Estratégia Adotada

Estratégias de procura **não informadas** usam apenas as informações disponíveis na definição do problema.

Nas estratégias de procura **informadas** dá-se ao algoritmo "dicas" sobre a adequação de diferentes estados.

O grupo decidiu implementar, nesta fase, mais três algoritmos de procura.

Ficamos então com dois de procura não informadas, **BFS** (Breath-First Search), já presente na primeira fase da realização do trabalho, e **DFS** (Depth-First Search), e também dois algoritmos de procura informadas, a procura **A*** e procura **Greedy/Gulosa**.

A figura seguinte apresenta a solução ótima e o seu custo, resultado da procura DFS implementada, para o mapa gerado referente à **Figura 2** (*Circuito aleatório de dimensão 10x10*). Os resultados foram bastante maus.

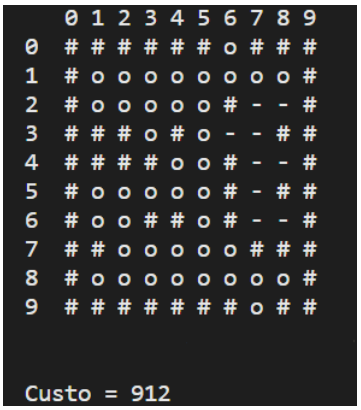

Figura 5: Procura DFS

A figura seguinte apresenta a solução ótima e o seu custo, resultado da procura BFS implementada, para o mesmo mapa. Os resultados foram satisfatórios.

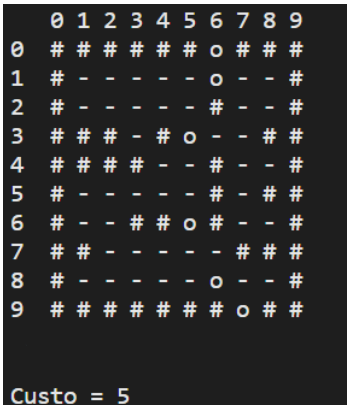

Figura 6: Procura BFS

A figura seguinte apresenta a solução ótima e o seu custo, resultado da procura A* implementada, para o mesmo mapa. Os resultados foram satisfatórios e idênticos à procura BFS.

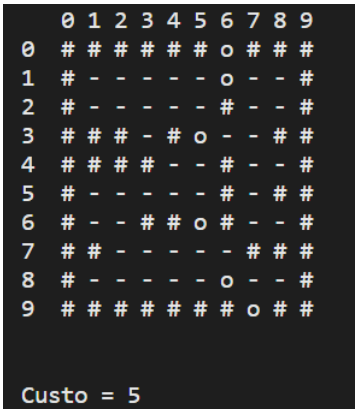

Figura 7: Procura A*

A figura seguinte apresenta a solução ótima e o seu custo, resultado da procura Greedy/Gulosa implementada, para o mesmo mapa. Os resultados não foram os melhores.

Figura 8: Procura Greedy/Gulosa

Depois de analisarmos todos os resultados, os melhores resultados foram das procuras BFS e A*. As outras procuras não são as mais adequadas para resolver problemas deste tipo.

8 Modo Competitivo

Nesta segunda fase do trabalho, foi nos proposto adicionar uma opção em que se pudesse entrar num tipo de competição entre, pelo menos, dois participantes.

No Menu (apresentado anteriormente na Figura 1), escolhendo a opção número 7 é possível entrar nesse *Modo Competitivo*, onde introduzimos qual o tipo de procura que cada um dos dois participantes irão utilizar para chegar à meta e, seguidamente, é desenvolvido o caminho de cada um e apresentado, finalmente, o custo de cada um. Naturalmente, o que conseguir acabar o circuito com o menor custo é declarado o vencedor.

Nas figuras seguintes demonstramos um exemplo de um mapa e de como este modo competitivo se processa no terminal para esse mapa e como o utilizador introduz as suas opções.

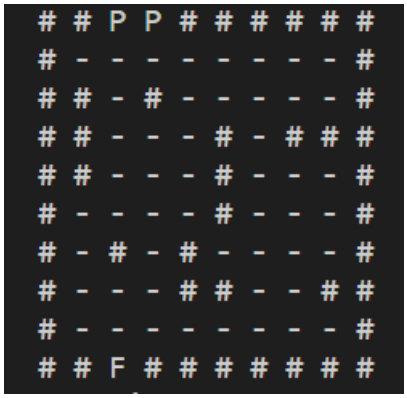

Figura 9: Mapa utilizado para o Modo Competitivo

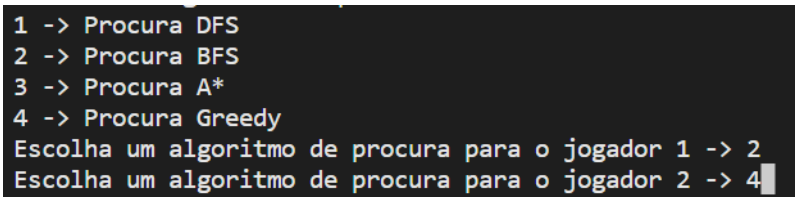

Figura 10: Seleção dos algoritmos de procura para cada jogador

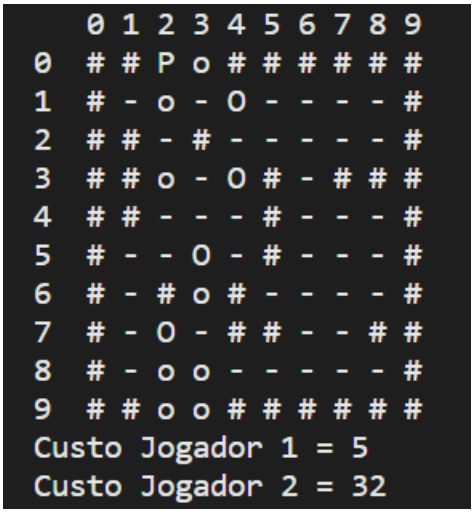

Figura 11: Resultado do Modo Competitivo

Analisando a Figura 11, conseguimos concluir, facilmente, que o jogador 1 (procura BFS), representado por "O", ganhou ao jogador 2 (procura Greedy), representado por "o", pois o seu custo foi menor.

9 Conclusão

O grupo está satisfeito com a realização deste trabalho e tudo que aprendeu com a mesma. Conseguimos consolidar o que aprendemos nas aulas de Inteligência Artificial e aprofundar esses mesmos conhecimentos na linguagem ***python*** e na resolução de problemas através da conceção e implementação de algoritmos de procura.

Para além do que a Unidade Curricular nos tem ensinado a nível técnico, também melhorou o grupo a ultrapassar dificuldades a nível de trabalhar em grupo, organizar tarefas e superar outras dificuldades inerentes à realização de um trabalho deste tipo.