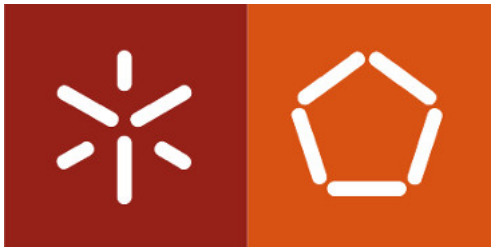

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Inteligência Artificial

Ano Letivo de 2022/2023

Instrumento de Avaliação em Grupo

Grupo 4

A96326 Bernard Ambrosio Georges
A97257 João Miguel Ferreira Loureiro
A94870 Rafael Picão Ferreira Correia
A91682 Robert Beniamin Szabo

23 de dezembro de 2022

Índice

1 Introdução

Este relatório é referente á segunda fase do Instrumento de Avaliação em Grupo da Unidade Curricular de Inteligência Artificial. Nesta fase final do projeto, implementamos **mais algoritmos de procura** no jogo de corridas VectorRace, uma **componente "multiplayer"** (vários participantes) e uma renovação do menu.

Nota: Este relatório é uma continuação do relatório da primeira fase, onde é descrita a base do projeto, entre outros assuntos, sendo essencial para a compreensão total deste relatório.

2 Novas Funcionalidades e Implementação

2.1 Menu

Na fase anterior, o menu era apresentado e utilizado através do terminal. De forma a facilitar e de melhorar a experiência de utilização do programa decidimos utilizar a biblioteca *Pygame* para construir uma interface gráfica onde o utilizador interagir com o rato de forma a escolher as opções pretendidas.

Figura 1: Menu principal do programa

O utilizador escolhe o circuito que pretende utilizar, podendo este ser um dos incluídos no programa (descritos no relatório da fase 1) ou, se preferir, um da sua própria criação. Para tal, apenas precisa de inserir o caminho do ficheiro *.txt* do mapa na caixa de texto e clicar no botão *"Adicionar circuito"*.

Após selecionado o circuito, é lhe apresentado uma representação gráfica do mapa escolhido, onde o utilizador pode então selecionar qual dos algoritmos implementados pretende utilizar.

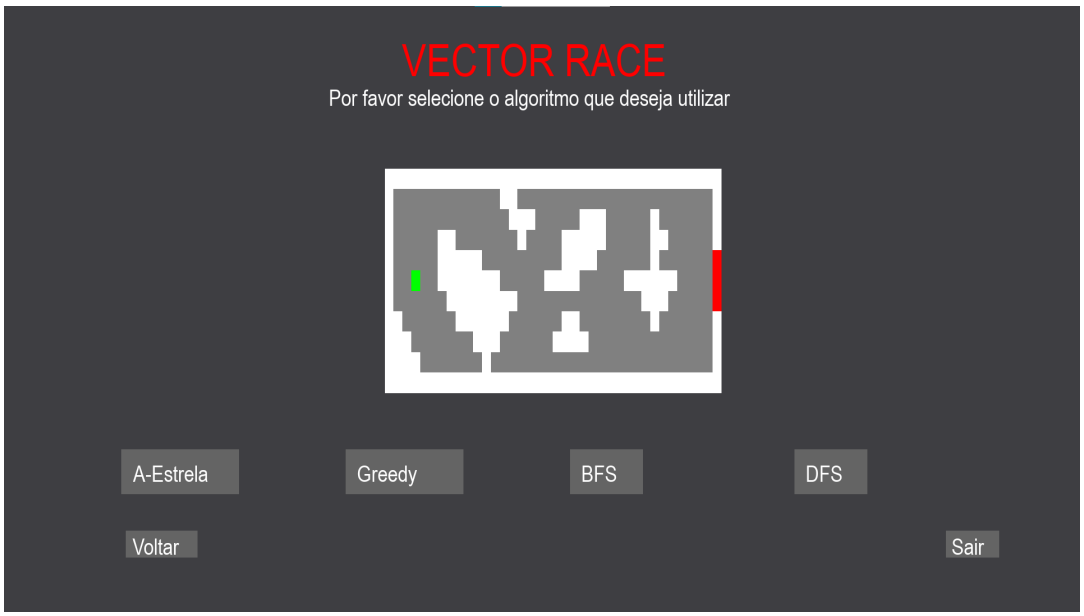

Figura 2: Seleção do algoritmo

É então apresentada o caminho e o custo da solução encontrada, podendo o utilizador escolher ver o grafo criado, ver o percurso (para poder ver o caminho passo a passo) ou voltar atrás e selecionar outro caminho.

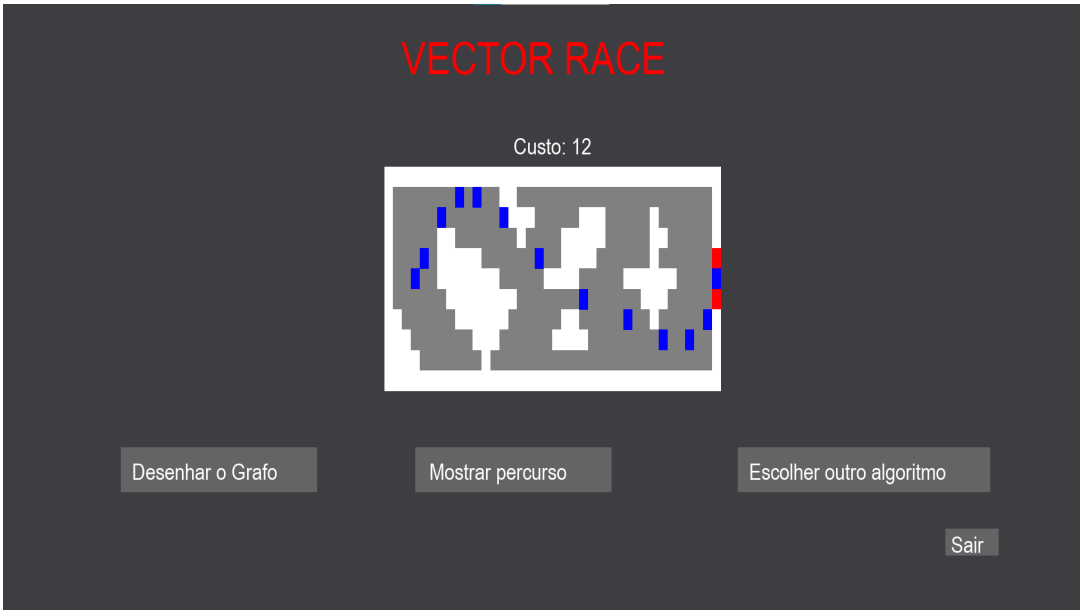

Figura 3: Menu dos resultados

2.2 Logs

Um dos requisitos que foram pedidos nesta fase consistia na **implementação da apresentação do caminho percorrido ao longo da execução do algoritmo**, isto é, todos os nodos expandidos/visitados.

Para este fim, á medida que o algoritmo é executado, são armazenados todos os nodos visitados que, no final da execução deste, são escritos no ficheiro *logs* encontrado na pasta *logs* segundo a ordem [[V],['C']], onde V são todos os nodos visitados e C o caminho da solução encontrada.

O ficheiro *logs* é apenas atualizado quando o utilizador escolhe a opção *"Mostrar percurso"* no menu do resultado do algoritmo.

Nota: Devido á alta complexidade dos grafos gerados e a possível quantidade enorme dos nodos visitados, é recomendado que seja apagado o ficheiro *logs* no fim da utilização do programa.

2.3 Modo Multiplayer

Outro dos requisitos pedidos nesta fase envolve a **implementação da possibilidade de ter pelo menos 2 participantes numa corrida.**

De forma a implementar esta funcionalidade corretamente, decidimos tomar as seguintes considerações para os algoritmos:

- outros participantes devem ser considerados obstáculos.
- um participante pode ocupar o mesmo espaço que outro, ao mesmo tempo ou não (pode passar por um espaço onde já tenha passado outro), tentando evitar o primeiro caso ao máximo.
- tentar ao máximo evitar colisões entre movimentos entre coordenadas, isto é, ao passar de uma coordenada para outra.

De forma a tentar evitar as possíveis colisões, implementamos um sistema de prioridade onde um jogador N tem prioridade sobre os jogadores cujo número vem depois (o 1 sobre 2,3,4,..., o 2 sobre 3,4,..., por exemplo).

Assim, **dependendo da ordem em que os algoritmos são escolhidos, obteremos caminhos e custos diferentes.**

A nossa implementação permite a ocupação de uma casa por dois participantes. Regra geral, ela **evita ao máximo este acontecimento** com sucesso. No entanto, devido ao sistema de prioridades, existem algumas combinações de algoritmos onde em algumas situações (por

exemplo, em percursos com caminhos estreitos) ocupam a mesma posição. Decidimos manter esta possibilidade devido á imprevisibilidade das situações, mas também porque permite observar que alguns algoritmos encontram a mesma solução apesar das filosofias de expansão diferentes.

De forma a ativar a componente multiplayer, basta **acrescentar mais posições de partida ('P') ao ficheiro do mapa**. O programa já vem com um circuito multiplayer incluído, o Circuito Multiplayer, contendo 4 participantes. Este circuito, derivado de um outro, apresenta vários caminhos distintos e diferentes de chegar á meta, o que permitiu testar e analisar melhor os caminhos escolhidos pelos participantes.

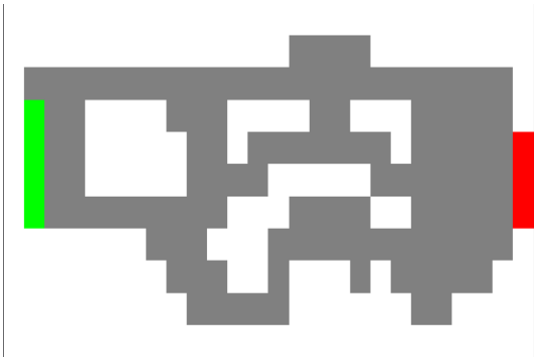

Figura 4: Circuito Multiplayer

O programa, detetando que existe mais do que uma posição de partida, apresenta ao utilizador a possibilidade de escolher qual algoritmo cada participante irá utilizar para percorrer o circuito.

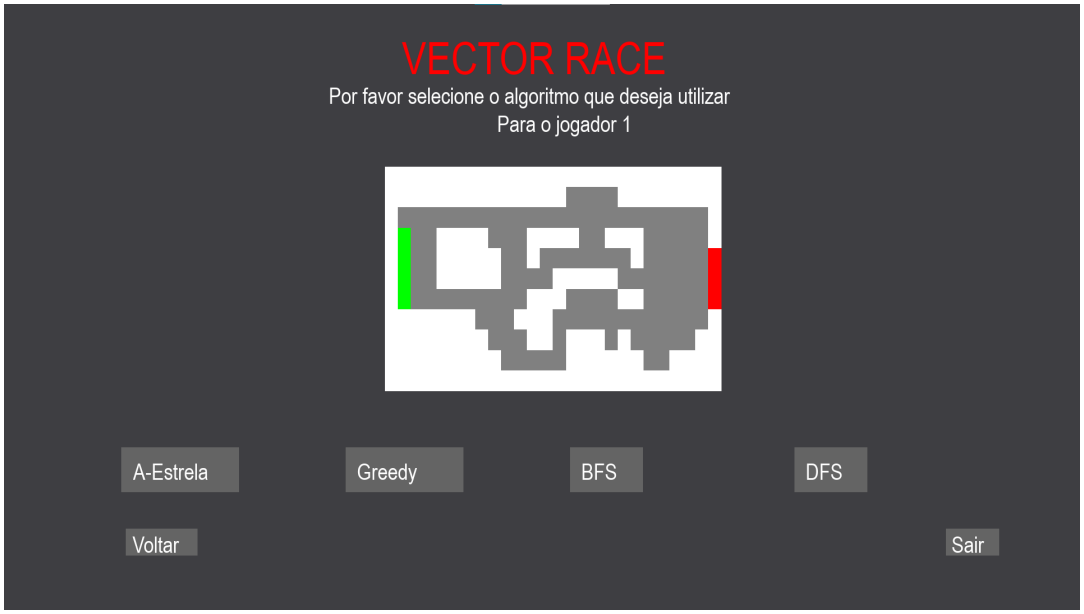

Figura 5: Menu multiplayer

É então apresentado o custo e o caminho percorrido por cada participante, podendo o utilizador ver evolução da corrida passo a passo. É atribuída uma cor a cada participante, de forma a visualizar o caminho percorrido por cada, notando que o programa atribui cores diferentes, no máximo, a 6 jogadores, sendo a partir do sétimo atribuída a mesma cor.

Nota: Apesar de poder inserir vários participantes, não conseguimos assegurar o funcionamento completamente correto quando existir uma quantidade enorme destes, sendo a quantidade recomendada de participantes 4.

2.4 Algoritmo AEstrela

O algoritmo de procura informada **AEstrela (A*)**, além de ser um dos mais fiáveis, é dos mais indicados para encontrar o caminho mais rápido entre dois pontos.

De uma forma muito resumida, o A* estima o custo a partir do nodo de um grafo onde se encontra até o nodo de chegada, utilizando uma heurística e evitando sempre expandir os caminhos mais dispendiosos. Apresenta, no entanto uma complexidade enorme, já que mantém todos os nodos na sua memória.

Como é um algoritmo de procura informada, o A* necessita de uma heurística. Neste problema, como sabemos as coordenadas do Estado Objetivo, decidimos **usar como heurística a distância entre o nodo atual e o nodo que correspondente ás coordenadas do objetivo**.

O algoritmo calcula as várias distâncias dos caminhos possíveis, atribuindo a cada um um peso, por sua vez equivalente á soma dos custos dos espaços percorridos.
Se encontrar um obstáculo/fora da pista num dado caminho, o custo desse caminho é multiplicado por 25 (penalização). O algoritmo escolhe então o caminho menos penalizador, evitando assim as colisões.

Utilizamos esta heurística de forma a otimizar a procura do caminho ótimo. Ao comparar a versão final com uma versão anterior do algoritmo, onde a heurística era só o custo de cada espaço, pode-se observar que o caminho escolhido passou possuir um custo menor:

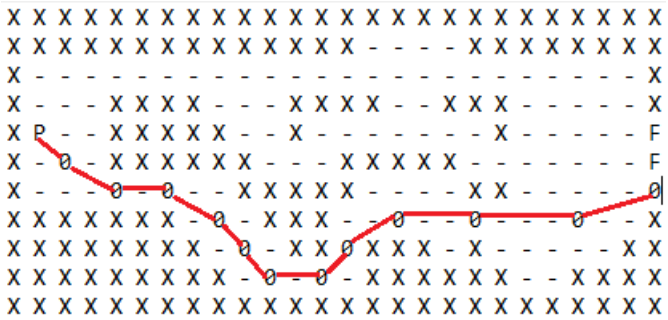

Figura 6: Percurso escolhido com a heurística anterior

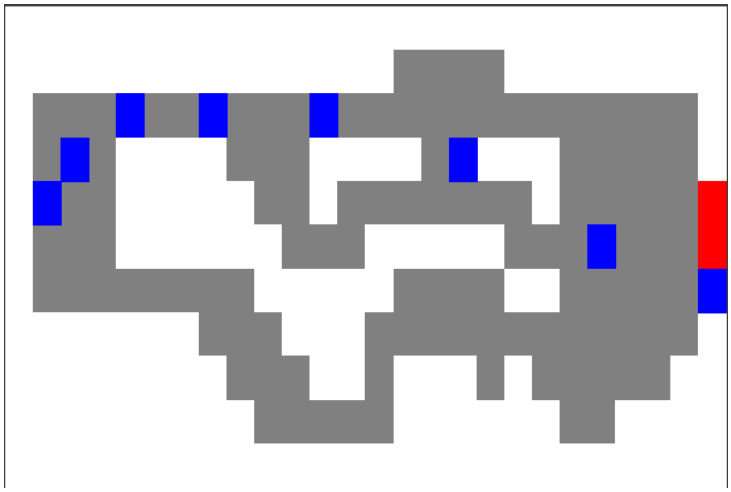

Figura 7: Percurso escolhido com a heurística atual

Um fator que tivemos que considerar na criação do grafo foi o caminho que o carro percorre entre os seus estados. Por exemplo, se entre dois estados (posições) válidos existir um caminho obstruído, essa sequência de estados deixa de ser um caminho válido.

Assim, de forma a resolver esse problema, o algoritmo prevê e esses casos através de uma segunda procura informada, onde o carro apenas move um passo de cada vez.

2.5 Algoritmo Greedy/Pesquisa Gulosa

O **algoritmo Greedy (Pesquisa Gulosa)** é uma algoritmo de procura informada que procura fazer a escolha o mais ótima possível em cada estado, ou seja, que expande o nó do gráfico que mais perto parece estar da solução.

Apesar de não encontrar sempre a melhor solução, costuma devolver uma solução razoável apesar de apresentar uma complexidade alta que pode ser atenuada pela a sua heurística.

Como heurística, utilizamos a mesma utilizada no A*, sendo esta **a distância entre o nodo atual e o nodo que correspondente ás coordenadas do objetivo**, com as mesmas caraterísticas. Decidimos usar a mesma pelos os mesmo motivos e para facilitar a comparação direta entre os dois.

Para a implementação deste algoritmo, decidimos fazer-lo de forma recursiva, semelhante ao método explorado nas aulas práticas. É interessante de notar que este algoritmo consome menos memória do que o A*, provavelmente devido ás diferentes implementações dos dois (nomeadamente a não recursividade do A*).

2.6 Algoritmo BFS

O **algoritmo BFS (Primeiro em Largura)**, como o nome indica, expande a sua procura ao explorar todos os nodos adjacentes de um grafo, podendo assim não encontrar o caminho ótimo. Não possui uma heurística devido ao seu caráter não-informado e a sua complexidade depende diretamente do tamanho do grafo.

Na nossa implementação, usa o mesmo grafo que o algoritmo A*, percorrendo o grafo ao nível da profundidade. Começa por explorar os nodos filhos do nodo de partida, explorando depois os filhos destes, e assim sucessivamente até encontrar um caminho até um dos nodos finais.

2.7 Algoritmo DFS

O **algoritmo DFS (Primeiro em Profundidade)** expande a sua procura ao expandir os nós de um ramo, em profundidade, passando apenas para os outros ramos depois de chegar ao fim do ramo. É não informado, não possuindo assim uma heurística, tendo uma complexidade que depende do número de nodos que explorar.

É considerada uma boa opção para casos de múltiplas soluções, podendo, possivelmente, ser uma boa opção para este projeto, mas nunca irá devolver a solução ótima, visto que devolve a primeira que encontrou.

Em termos de implementação, decidimos usar uma espécie de *stack*, de forma a controlar o caminho a percorrer. Essencialmente, ao chegar ao fim de um ramo e verificar que não chegou á meta, executa um **pop** dos nodos irrelevantes ao caminho, ficando na *stack*, no fim da procura, apenas o caminho da solução. Não é, portanto, feita de forma recursiva.

3 Análise dos Resultados

Para determinar o algoritmo ótimo, decidimos analisar e comparar os resultados fornecidos por cada um dos 4 algoritmos.

No caso apresentado, cada algoritmo **foi testado individualmente no circuito 2 e 3**, e consequentemente, **em simultâneo em uma corrida multiplayer no circuito Multiplayer**. Testes adicionais foram executados, sendo este caso escolhido devido a refletir, em geral, os resultados encontrados.

Assim, obtivemos os seguintes resultados:

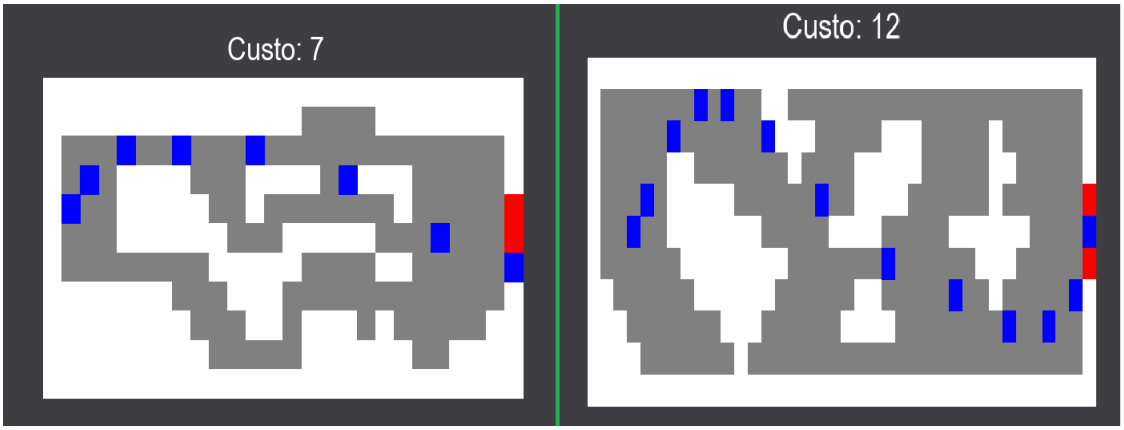

Figura 8: Resultado da aplicação do algoritmo A* no circuito 2 e 3

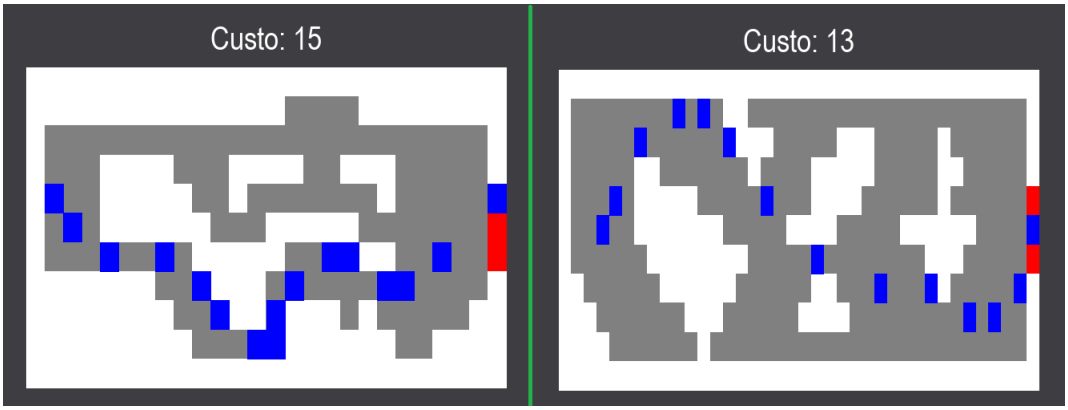

Figura 9: Resultado da aplicação do algoritmo Greedy no circuito 2 e 3

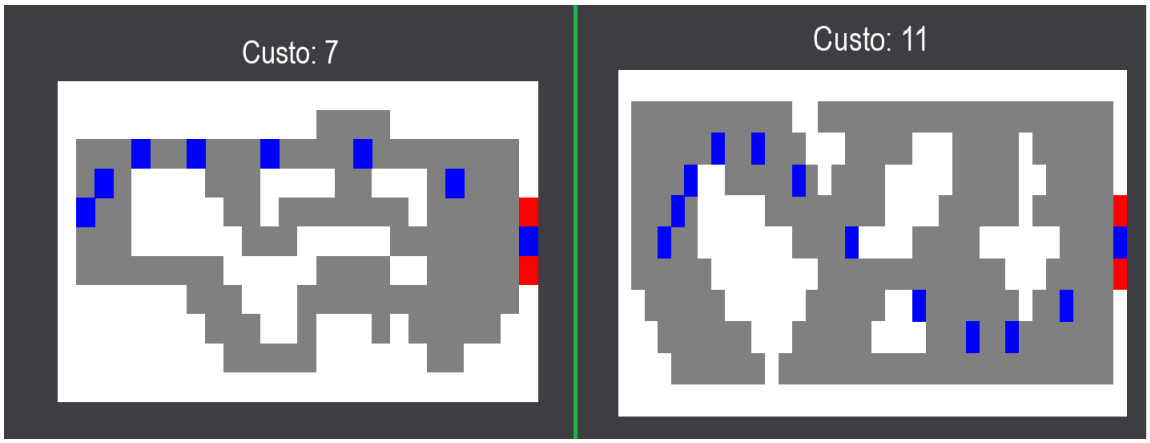

Figura 10: Resultado da aplicação do algoritmo BFS no circuito 2 e 3

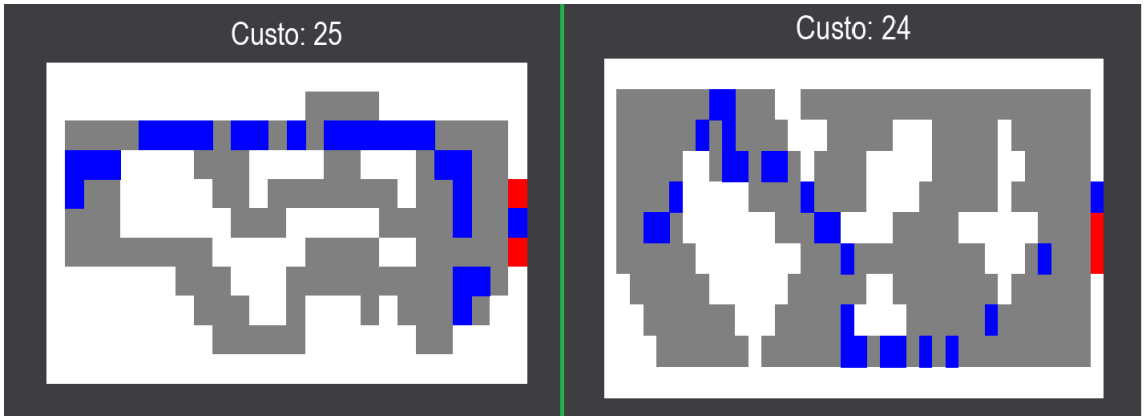

Figura 11: Resultado da aplicação do algoritmo DFS no circuito 2 e 3

Analisando os resultados individuais em primeiro lugar, podemos observar que **ambos os algoritmos A* e BFS encontraram as soluções menos dispendiosas no circuito 2** (custo 7), **tendo o BFS encontrado um caminho menos dispendioso por uma unidade no circuito 3** (custo 11 e 12, respetivamente).

De seguida vem o **algoritmo Greedy, com sensivelmente o dobro do custo no circuito 2** (custo 15) e com **uma solução ligeiramente mais dispendiosa que a do A* no circuito 3** (custo 13). Finalmente, temos o **algoritmo DFS, com um custo significativamente maior que os outros**, tendo um custo de 25 e 24 para o circuito 2 e 3, respetivamente.

Estes resultados estão dentro dos previstos, tendo em conta as semelhanças nas implementações do A* e do BFS, e as filosofias de expansão de cada um dos algoritmos, que previa o A* como um dos mais indicados para este problema e o DFS o caminho menos eficiente e mais estranho.

É também de notar que o algoritmo Greedy foi o único que escolheu construir a sua solução a partir do caminho inferior, apesar de partilhar a mesma heurística com o A*, muito provavelmente devendo-se á sua implementação recursiva e á sua posição de partida.

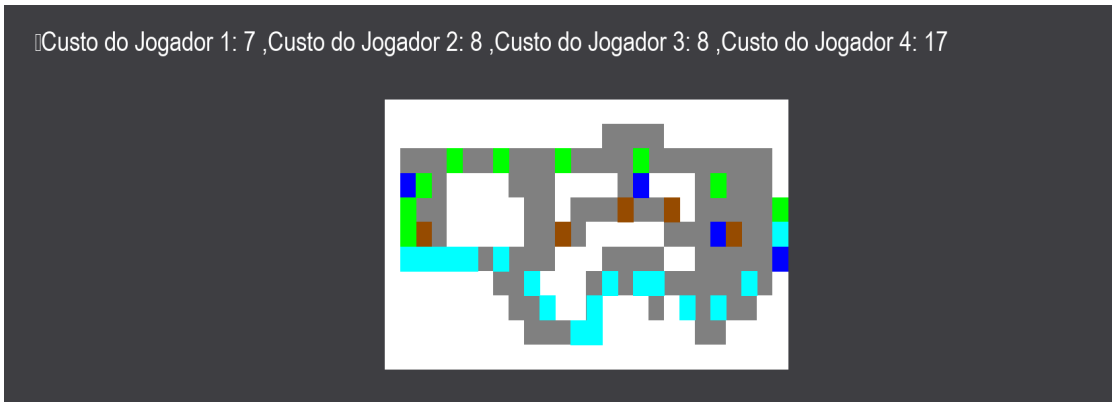

Figura 12: Resultado da corrida multiplayer

Olhando agora para os resultados no multiplayer, o caminho da cor azul (1) é do algoritmo A*, a da cor castanha (2) o Greedy, a do verde (3) o BFS, e por ultimo, a azul-claro (4) o DFS.

Nota: Devido a alguns participantes passarem pela a mesma posição, no mapa pode parecer que alguns caminhos estão cortados (por exemplo, se o azul e o castanho passarem na mesma posição, apenas um dos dois será visível). No entanto, na demonstração do caminho passo a passo, dá para verificar o caminho que cada participante faz, sendo por exemplo, que um passou depois de outro.

Observando a corrida em ação, pode-se verificar que **o vencedor da corrida foi o A***, em segundo houve um empate entre o Greedy e o BFS, e em último ficou o DFS. Em termos de custo, o A* foi o que encontrou o caminho de menor custo (7) , seguido pelo Greedy e o BFS, que tiveram o mesmo custo (8) com uma diferença ligeira do anterior. Já o DFS foi o que encontrou o caminho mais dispendioso, com custo 17, muito superior aos anteriores.

Em geral, **cada algoritmo escolheu uma direção diferente a percorrer**,podendo observar que os algoritmos de procura informada decidiram usar como ponto de partida o caminho de cima e os não informados o de baixo, divergindo todos depois para os seus respetivos caminhos.

4 Resultados Finais

Algoritmo	Custo Individual (Circuito 2)	Custo Individual (Circuito 3)	Custo Multiplayer	Posição Multiplayer	Total
A*	7	12	7	1º	26
Greedy	15	13	8	2º	36
BFS	7	11	8	2º	26
DFS	25	24	17	4º	66

Figura 13: Sumário dos resultados

Tendo em conta os resultados acima observados, podemos concluir que, em geral, **o melhor algoritmo para o VectorRace é o A***, com uma menção honrosa para o BFS. Além de serem os menos dispendiosos, mostram-se também os melhores participantes na corrida com os outros algoritmos, ficando em primeiro e segundo lugar, respetivamente. É de notar, no entanto, que o A* é o algoritmo mais consistente dos dois, sendo por isso, considerado o melhor.

Já **o pior algoritmo, revelou-se ser o DFS**, devido ás soluções dispendiosas que encontra sistematicamente e á sua má prestação na corrida multiplayer. Este resultado não é devido a uma má implementação, mas sim devido **á filosofia de procura deste algoritmo, claramente visível nas soluções que encontrou**.

Estes resultados foram esperados, já que foram de acordo com as suas caraterísticas abordas nesta UC, nomeadamente na superioridade, em geral, dos algoritmos de procura informada em

relação aos não informados.

A **proximidade entre os resultados do BFS e do A* é notável**, podendo no entanto estar relacionada aos mapas usados para testar ou com a nossa implementação, que pode favorecer alguns algoritmos no lugar de outros.

5 Conclusão

Através do desenvolvimento deste trabalho, conseguimos aplicar os conhecimentos que obtemos nas aulas desta UC, nomeadamente, a criação de algoritmos úteis ao desenvolvimento de um inteligência artificial.

Ao comparar as soluções encontradas pelos diferentes algoritmos, conseguimos entender melhor as vantagens e desvantagens de cada um, juntamente com a filosofia que cada um implementa para alcançar o seu objetivo, o que nos proporcionou um melhor entendimento dos conceitos lecionados da cadeira.

Conseguimos também aprofundar o nosso conhecimento da linguagem Python que levaremos para o nosso futuro académico e profissional, dada á utilidade desta.

Em termos de aspetos que gostaríamos de ter melhorado temos a *performance* do programa e eventuais otimizações dos algoritmos possamos ter deixado por fazer.

Em geral, **estámos satisfeitos com o resultado final** e terminamos este projeto com mais experiência neste ramo da Engenharia Informática que, sem dúvidas, nos irá auxiliar em futuros projetos.