# Inteligência Artificial

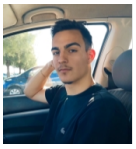
## Relatório do Trabalho Prático - Fase 2

LEI - 2022/2023

## Descrição do Desenvolvimento e Resultados Obtidos

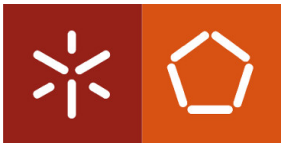
Grupo 15


Rui Chaves A83693 

Maurício Pereira A95338 

Afonso Bessa A95225 

Martim Ribeiro A96113 


*Universidade do Minho*

# Índice

## Lista de Figuras

# 1 Introdução

No âmbito da Unidade Curricular de **Inteligência Artificial** foi proposto o desenvolvimento, conceção e implementação de diversos Algoritmos de Procura para a resolução do jogo *VectorRace*.

O seguinte relatório irá expressar todas as alterações efetuadas à Fase 1, retratar e exibir tomadas de decisão na implementação do Ambiente Competitivo e dos Algortimos, bem como, todos os acontecimentos que levaram à resolução das etapas relativas a esta Fase 2.

## 1.1 Ponto de Partida

Antes de iniciar a Fase 2, o grupo de trabalho decidiu contactar os docentes de maneira a esclarecer todas as dúvidas que ficaram pendentes na fase anterior e incertezas face à segunda fase. Assim, após uma reunião presencial foi possível:

- Primeiramente, esclarecer a precisão e acerto do Algoritmo de Largura, bem como, clarificar como adaptar os Circuitos para o tamanho ideal para a Fase2.
- Em segundo lugar, dilucidar como implementar a velocidade e heurística nos Algortimos *A** e *Greedy* e, por fim, definir possíveis maneiras de tratar o problema de colisão de Jogadores no Ambiente Competitivo.

**Palavras-Chave:**

IA, *Python*, Ambiente Competitivo, *A**, *Greedy*, Colisão, Heurística.

**Acrónimos e Siglas**

- **BFS** ⇒ Breadth-First Search
- **DFS** ⇒ Depth First Search

## 2 Estrutura do Projeto

O Projeto encontra-se partido em onze ficheiros *.py*, incluindo também uma pasta Circuitos onde se encontram todos os *CircuitosXX.txt*.

A maioria dos Ficheiros já foram abordados ao promenor no relatório da Fase1, pelo que, de forma a evitar repetições, apenas nos iremos referir às alterações feitas para esta Fase2.

Na parte da representação gráfica do *VectorRace*, foi criado um novo ficheiro, o *vectorrace.py*, que funciona de forma muito semelhante aos outros dois ficheiros *VectorRace* (*vectorrace_ler.py* e *vectorrace_criar.py*) mas a principal diferença baseia-se nos argumentos recebidos. Recebe os dois Algoritmos escolhidos pelos Jogadores e desenha graficamente os seus caminhos e os seus custos, bem como, ponto de partida, meta, checkpoints, possíveis colisões e legenda dos mesmos.

A pasta Circuitos contém agora os Circuitos da fase prévia, simplesmente com a mudança da identificação '_c' significando se o Circuito tem ou não *checkpoint* e uma pasta *Multiplayer* que contém todos os cinco Circuitos redimensionados e adaptados, ou seja, com duas partidas e uma meta, para o Ambiente Competitivo.

As mudanças mais relevantes e expressivas aconteceram nos ficheiros:

- Graph.py
- Race.py
- Main.py

No primeiro ficheiro foram acrescentadas todas funções relativas aos novos algoritmos **A*** e **Greedy** e as suas funções para calcular a heurística (*heuristica_aStar* e *heuristica_greedy*).

De seguida, no Ficheiro *Race.py* foi simplesmente acrescentada a função para verificar a ocorrência de colisões (*check_collision*) e as suas auxiliares.

Por fim, no *main*, foi acrescentado ao menu apresentado na fase anterior a possibilidade do Ambiente Competitivo entre dois Jogadores (tópico abordado em mais promenor de seguida).

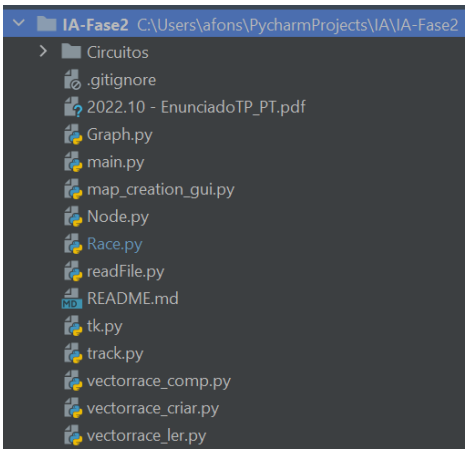



**Figura 1.** Ficheiros Do Projeto

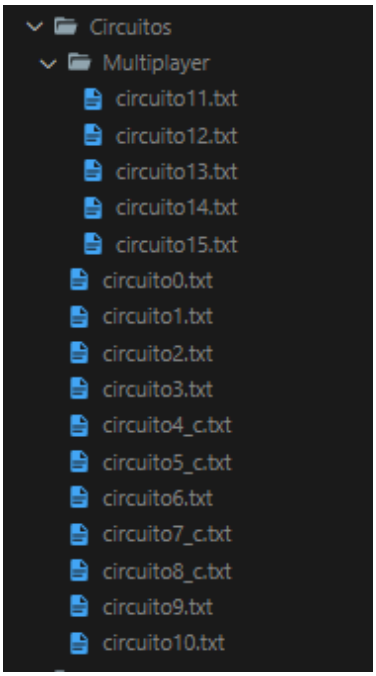



**Figura 2.** Circuitos Disponíveis

## 2.1 Manual de Instrução

Para executar o nosso código e tirar proveito de todas as suas funcionalidades basta usar o comando:

**python3 main.py**

Após isto, irá aparecer como *output* um Menu que, através da introdução de números, tornará possível navegar entre cada um dos seus *submenus*, voltar ao passo anterior, voltar ao menu principal, terminar processo, bem como, experimentar o Ambiente Competitivo, entre outras várias funcionalidades.

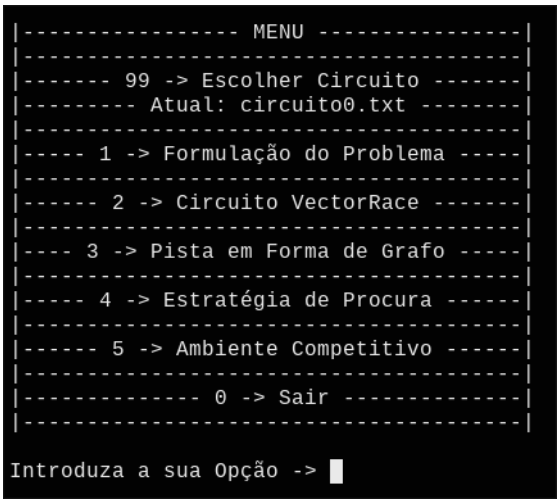



**Figura 3.** Menu Inicial

# 3 Avanços relativos à Fase 1 e problemas associados

Tal como já foi mencionado anteriormente, depois da reunião com o Docente, o foco de toda a equipa voltou-se para a implementação da velocidade em cada Circuito.

Após a tentativa de implementar a velocidade através do *pre-processing* dos nodos, fazendo a construção do grafo associando a cada nodo todos os nodos adjacentes a uma distância Manhattan de v dinâmica, recorrendo a um *array* bidimensional de velocidades, surgiu-nos um problema: não sabíamos qual a velocidade que devíamos admitir para o cálculo da posição seguinte, pois em cada ponto intermédio do Circuito podíamos ter duas ou mais possibilidades com a qual chegamos a esse nodo intermédio, com antecessores distintos. Por exemplo, para nos movimentarmos para um nodo (4,7) temos as opções:

Posição Anterior , Velocidade Associada

((4,6), (0,1)) ; ((4,5),(0,2))

Ou seja, o nodo anterior ser (4,6) e vir com velocidade (0,1) ou o nodo anterior ser (4,5) e velocidade associada ser (0,2). Assim, quando avançássemos, chegávamos a uma indecisão sobre qual das velocidades usar, de modo a manter a fiabilidade do Algoritmos. Foi discutido assumir sempre como velocidade utilizada no calculo a 1ª da lista, mas isto, apesar de aparentar dar uma solução correta aquando da execução de um algoritmo para mapas pequenos, limitava invariavelmente a expansao dos caminhos possiveis. Após nos apercebermos da complexidade das nossas dúvidas, decidimos contactar novamente o mesmo Docente.

No decorrer da segunda reunião, após expormos todas as nossas dúvidas, o professor sugeriu usarmos uma média ponderada das velocidades, algo que imediatamente não nos agradou, uma vez que não iria manter a igualdade de escolha de nodos e veracidade dos Algoritmos. Após mais algum debate de ideias, o Docente percebeu que a nossa dúvida principal encontrava-se na compreensão dos Algoritmos $A*$ e *Greedy* e da sua heurística respetiva e na mistura disso com a velocidade. Após uma breve explicação dos Algoritmos com

recurso a exemplos, os elementos do grupo ficaram totalmente esclarecidos e prontos para modificar os Algoritmos, as suas heurísticas e a construção do grafo.

Durante esse tempo criamos novos Circuitos, com recurso à ferramenta criada na fase anterior, permitindo uma melhor gestão do tempo, de modo a possuirem um tamanho ideal, bem como, duas partidas e uma meta, tal como foi mencionado anteriormente.

Por fim, no momento de entrega da primeira fase, julgávamos que um dos Algoritmos não Informados apresentado, mais precisamente o DFS, estava incorreto, pois a solução do caminho do Circuito por este apresentado afigurava-se para o grupo suspeita e questionável. Porém, após uma análise, o Professor, na primeira reunião, clarificou os membros que o Algoritmo se encontrava cem por cento correto. De seguida, iremos mostrar o resultado do Algortimo DFS a um Circuito:

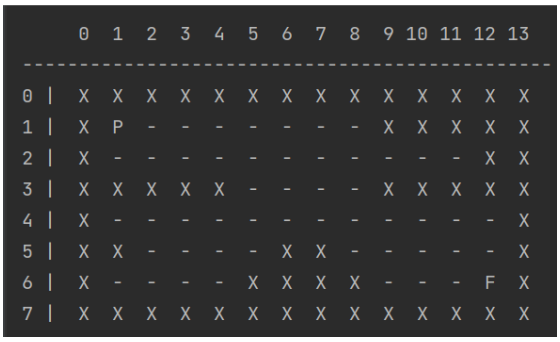



**Figura 4.** Circuito Inicial

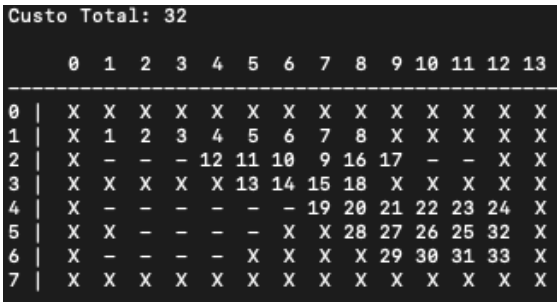



**Figura 5.** Resultado do Algoritmo DFS

# 4 Algoritmos Implementados

Os Algoritmos disponíveis para este Projeto estão divididos em dois subtópicos:

- Algoritmos não Informados:

  1) BFS
  2) DFS

- Algoritmos Informados:

  1) A*
  2) Greedy

De seguida vamos explicar cada um dos Algoritmos.

## 4.1 Algortimos não Informados

Algoritmos não Informados, também conhecidos como Algoritmos de Procura cega, são Algoritmos que exploram um espaço de procura sem usar informações sobre o estado atual ou o objetivo final.

Para estes Algoritmos "cegos" implementar velocidade seria entrar em contradicão, uma vez que iríamos estar a fornecer informações para a sua resolução. Logo a solução é feita vértice a vértice.

Adotamos velocidade fixa nestes Algortimos pois, como vai ser explicado adiante, a resoluçao é feita vértice a vértice, ou seja, o Algortimo não tem a capacidade de "saltar"barreiras.

**4.1.1 BFS** O Algoritmo BFS segue uma estratégia de procura em largura, explorando todos os vértices adjacentes ao vértice inicial antes de explorar vértices mais distantes.

A sua execução começa por um vértice inicial no grafo e, em seguida, através da exploração de todos os vértices adjacentes a esse vértice. Quando todos os vértices adjacentes ao vértice inicial tiverem sido explorados, o algoritmo passa para explorar os vértices adjacentes aos vértices já explorados, e assim por diante, até que todos os vértices do grafo tenham sido visitados.

**4.1.2 DFS** O Algoritmo DFS segue uma estratégia de ir a cada ramo do grafo até o seu final, antes de passar para o próximo ramo.

É executado começando por um vértice inicial no grafo e, em seguida, explorando todos os ramos a partir desse vértice. Quando não é possível prosseguir com a exploração de mais ramos a partir de um determinado vértice, o algoritmo retrocede para o último vértice que ainda tem ramos não explorados e continua a explorar a partir daí. Esse processo é repetido até que todos os vértices do grafo tenham sido visitados.

## 4.2 Algortimos Informados

Os Algoritmos Informados são uma classe de Algoritmos de Procura, que utilizam informações sobre o problema para orientar a busca pelo espaço de estados e encontrar a solução mais eficientemente. Ao contrário dos Algoritmos não Informados, os Algoritmos Informados recorrem a heurísticas ou informações específicas do problema para direcionar a procura e encontrar a solução de forma mais eficiente.

**4.2.1 A*** O Algoritmo A* é um Algoritmo de Procura que é utilizado para encontrar o caminho mais curto entre dois pontos num grafo. Utiliza uma função de avaliação que leva em conta, não apenas a melhor opção atual, mas também a distância estimada até a solução final. Isso garante que o caminho encontrado pelo Algoritmo é o caminho mais curto possível até a solução.

**4.2.2 A* - Heurística** Para a determinação deste elemento, calculamos o valor correspondente à distância na diagonal até ao nosso

ponto objetivo (meta) mais a soma do valor da aresta (que é sempre 1).

**Heurística = distância restante até ao objetivo + 1**

Idealmente, teríamos tido em consideração nesta heurística o valor da velocidade, mas infelizmente não a foi possível implementar, por termos passado muito tempo a pensar incorretamente no problema, tentando implementar a mesma na construção do grafo. As seguintes heurísticas também foram consideradas para implementação, apesar de não fazerem parte do resultado final:

1) h(n) = max_velocidade - velocidade_atual

2) h(n) = distancia_ate(focus_of_turn)

3) h(n) = possivelMover(x, y) ? 0 : infinito

O ponto 1) faria com que fossem preferidos caminhos associados a uma velocidade maior, o ponto 2) faria com que fossem explorados caminhos mais perto da borda interior de uma curva e, por fim, o ponto 3) faria com que fossem evitadas paredes.

**4.2.3 Cálculo de F(n)** De notar, que todas estas heurísticas poderiam ser consideradas para o cálculo do F(n). De salientar também que o valor da heurística de um nodo que não faça parte do mapa, isto é, no cálculo deste F(n) no algoritmo, admite-se como sendo 1000, um valor muito superior a F(n) dos nodos existentes no grafo.

Atualmente, o algoritmo A* calcula o valor de F(n) da seguinte forma: `F(n) = g(n) + h(n) * (1 + 1/1000)`.

A adição do fator `1 + 1/1000` advém de supormos que os caminhos não são superiores a 1000. O facto de definirmos este valor, põe em causa a "admissibilidade"da heurística, mas verificamos que o resultado desta adição ajudou a quebrar empates de heuristica e a fazer com que o algoritmo explorasse muito menos do mapa do que anteriormente.

**4.2.4 A*** Implementamos também um algoritmo A* que considera a velocidade de uma forma bastante básica.

O algoritmo recorre exatamente às mesmas heurísticas do algoritmo em cima descrito pelo que estas, de novo, não consideram a velocidade nem são dinâmicamente calculadas. Foi adicionado apenas uma variável inteiro que incrementa constantemente, feitas alterações nas funções calcula_custo - como o proximo nodo pode não ser adjacente ao atual, definimos 1 como o custo de ir para uma distancia v - e getNeighbours - retorna agora todos os nodos a uma distância de v, inteiro velocidade - e adicionada a função barriersBetween - basicamente define uma linha reta entre o nodo atual e o nodo para o qual queremos ir e testa se há barreiras -, testada no getNeighbours.

**4.2.5 Greedy** O Algoritmo Greedy é um algoritmo de Prcocura que é utilizado para encontrar a solução ótima para um determinado problema. Escolhe sempre a melhor opção disponível naquele momento, com base numa função de avaliação. Isso pode levar a um resultado ótimo, mas não é garantido.

Neste algortimo a heuristica consiste no custo estimado do caminho mais curto do estado n para o objetivo. Optamos pela criação de uma função que calculasse este custo, que é chamada aquando da resolução do Algortimo.

**4.2.6 Greedy - Heurística** Para a determinação deste elemento, que é atualizado a cada iteração, calculamos o valor correspondente à distância na diagonal até ao nosso ponto objetivo (meta).

**Heurística = distância restante até ao objetivo**

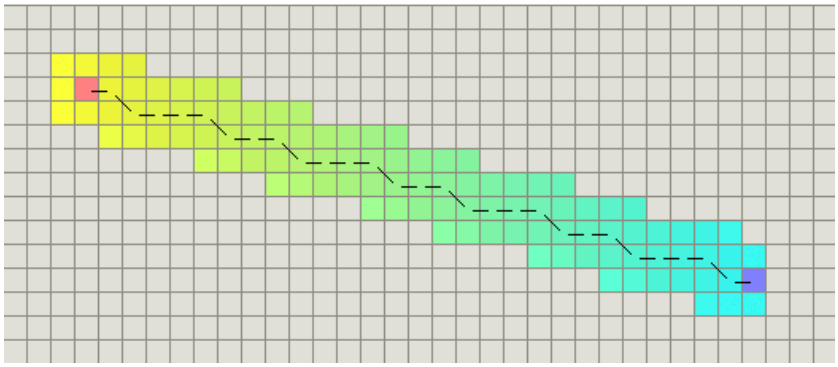


**Figura 6.** Representação da Distância Diagonal para ambos os Algoritmos

## 5 Ambiente Competitivo

Após uma leitura cuidadosa do Enunciado sobre a Fase2, mais precisamente, sobre o Ambiente Competitivo, o grupo pensou como poderia alterar e aprimorar o Menu anteriormente realizado de maneira a incluir a nova *feature*: **Ambiente Competitivo**.

Imediatamente, algumas ideias/alterações necessárias surgiram ao grupo. Algumas delas foram:

- Dois Jogadores;
- Escolher Colisão;
- Cada Jogador escolhe um Algoritmo de Procura;
- Cada Jogador tem um ponto de partida e meta associados;
- Existência de Colisão dos Jogadores.

Assim, foi criado um novo tópico no Menu principal, com a designação de Ambiente Competitivo que, após o Utilizador selecionar, surge um submenu para selecionar o Circuito que pretende, depois um novo submenu com a possibilidade de escolher a colisão: de forma aleatória ou tendo em conta o menor custo até ao momento. Após isto, são apresentados dois *submenus*: primeiramente o relativo ao Jogador1 e, posteriormente, relativo ao Jogador2, onde permite, para além das funcionalidades básicas de voltar atrás e sair, escolherem um Algoritmo de Procura Informada ou não Informada. Após os Algorimos escolhidos, um *VectorRace* surgirá com as duas soluções para cada Algoritmo com a devida correção caso uma colisão aconteça.

Através de uma legenda pode verificar-se a cor associada a cada Algoritmo e o seu respetivo custo.

Com esta nova *feature* instalada foi necessária a alteração do formato dos nossos Circuitos, passando a ter dois pontos de partida distintos, um para cada jogador, e uma meta comum aos dois. É possível, também, verificar as decisões de um mesmo Algoritmo a sair de pontos diferentes, bem como, a diferença de tomadas de decisão entre Algoritmos de Procura diferentes. Relembre-se a possibilidade de visualizar colisões entre jogadores.

Apesar de a representação gráfica dos circuitos já estar feita para fase anterior, esta nova atualização envolveu alguns obstáculos. Seria agora necessário acrescentar dois caminhos novos e possíveis colisões à implementação o que, numa fase inicial, não pareceu suscitar muita dificuldade. No entanto, durante a implementação, reparamos que os circuitos gráficos estavam a ser representados tendo em conta que as coordenadas do mesmo se encontravam no primeiro quadrante de um referencial cartesiano, enquanto que o resto do programa operava no segundo quadrante. Foi, por isso, necessário fazer algumas conversões de coordenadas.

Na Figura 7 é possível ver um exemplo de Ambiente Competitivo com dois Jogadores, um a amarelo e outro a azul, partindo de pontos diferentes assinalados a cor vermelha. A meta encontra-se assinada com a cor verde e existe, inclusive, uma colisão representada com a cor roxa. Por fim, as bordas e barreiras encontram-se representadas com a cor preta.

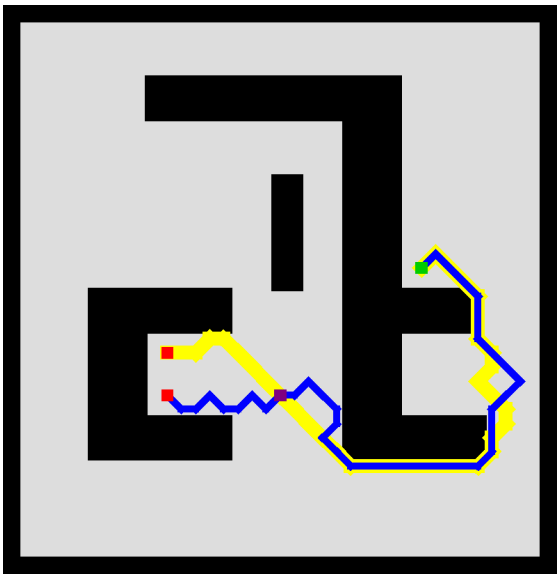


**Figura 7.** Modo Competitivo

De seguida iremos mostra a representação do Menu abordado anteriormente:

```
|----------------- MENU ----------------|
|---------------------------------------|
|------- 99 -> Escolher Circuito -------|
|--------- Atual: circuito0.txt --------|
|---------------------------------------|
|----- 1 -> Formulação do Problema -----|
|---------------------------------------|
|------ 2 -> Circuito VectorRace -------|
|---------------------------------------|
|---- 3 -> Pista em Forma de Grafo -----|
|---------------------------------------|
|----- 4 -> Estratégia de Procura ------|
|---------------------------------------|
|------ 5 -> Ambiente Competitivo ------|
|---------------------------------------|
|-------------- 0 -> Sair --------------|
|---------------------------------------|
```

**Figura 8.** Menu Inicial


**Figura 9.** SubMenu Colisão

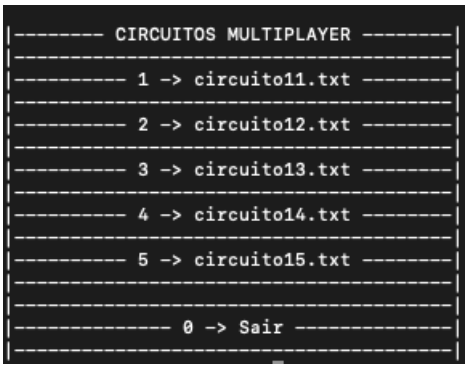



**Figura 10.** SubMenu Circuitos *Multiplayer*

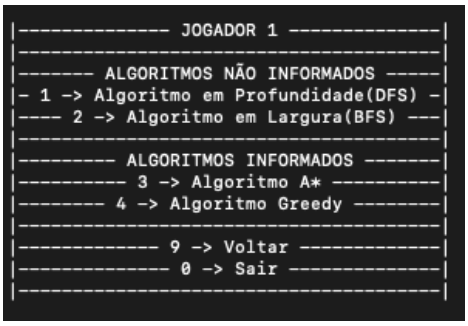



**Figura 11.** SubMenu Jogador 1

## 6 Colisão entre Algoritmos

Inicialmente, a primeira ideia para a resolução da Colisão no Ambiente Competitivo foi retroceder uma posição em cada jogador, isto é, passavam a ocupar a posição exatamente antes da colisão e voltar a recalcular o melhor caminho a partir desse ponto, simultane-

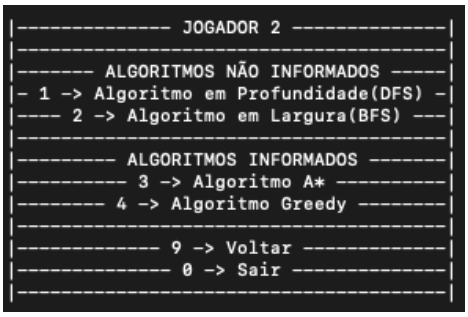



**Figura 12.** SubMenu Jogador 2

amente. De forma a evitar novas colisões no mesmo ponto (muito possívelmente, no que ocorreu a colisão prévia), o grupo decidiu deixar passar um dos jogadores em 1º lugar, seguido do outro. Assim, de forma a melhorar este aspeto, foram dadas as opções ao utilizador sobre que critério usar a propósito de qual jogador passa em primeiro lugar. As opções estão representadas na figura 9..
Deste modo, se houver colisão, podemos desempatar de forma aleatória ou, então, olhando para os custos até ao momento de cada Jogador, avançando em primeiro aquele cujo custo é menor.

De seguida, vamos apresentar uma foto com o exemplo de uma colisão a roxo.

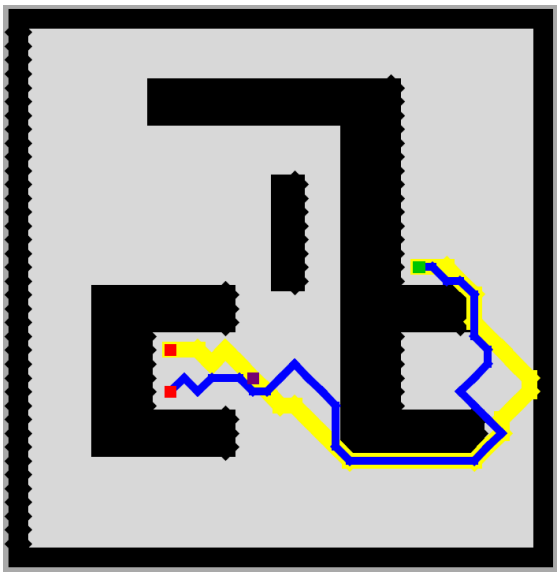


**Figura 13.** Ambiente Competitivo com Colisão entre A* e Greedy

# 7 Resultados

## 7.1 Resultados dos Algoritmos

Após realizarmos vários testes em diferentes Circuitos, decidimos escolher o *Circuito0.txt*, pois consideramos que é ele o que melhor permite visualizar a execução e observar diferenças entre os Algoritmos, bem como presença ou não de colisões entre Circuitos.

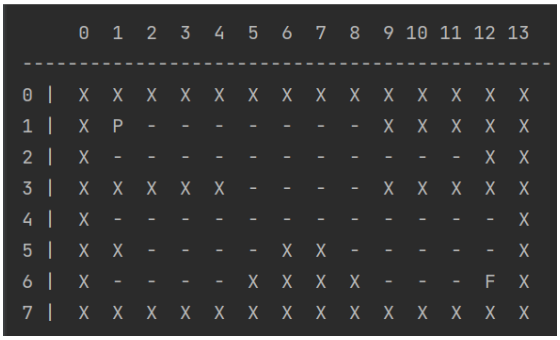



**Figura 14.** Circuito Escolhido

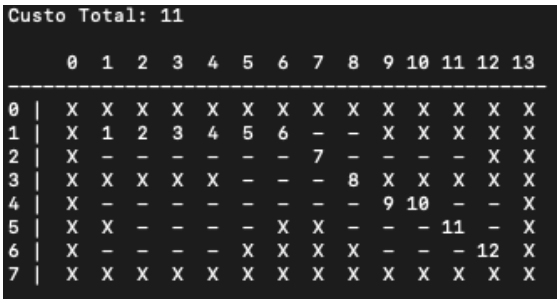



**Figura 15.** Resultado do Algoritmo BFS

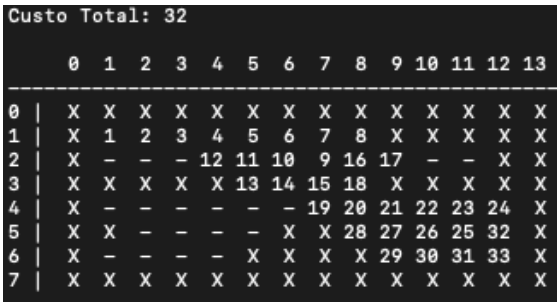



**Figura 16.** Resultado do Algoritmo DFS

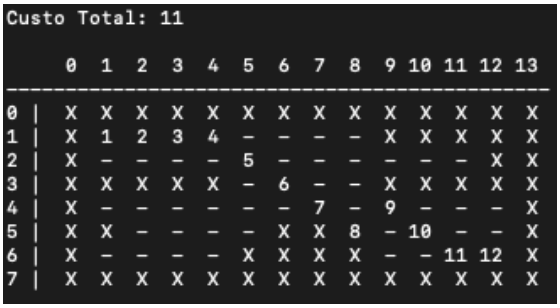



**Figura 17.** Resultado do Algoritmo A*

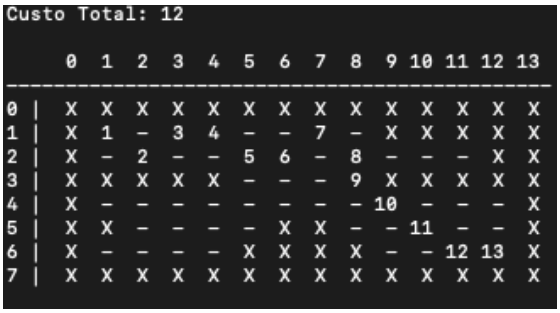



**Figura 18.** Resultado do Algoritmo Greedy

## 7.2 Resultados dos Algoritmos no Ambiente Competitivo

Após realizarmos vários testes em diferentes Circuitos para os diferentes Algoritmos, decidimos escolher o *Circuito11.txt*, pois consideramos que é ele o que melhor permite visualizar a execução e observar diferenças entre os Algoritmos.

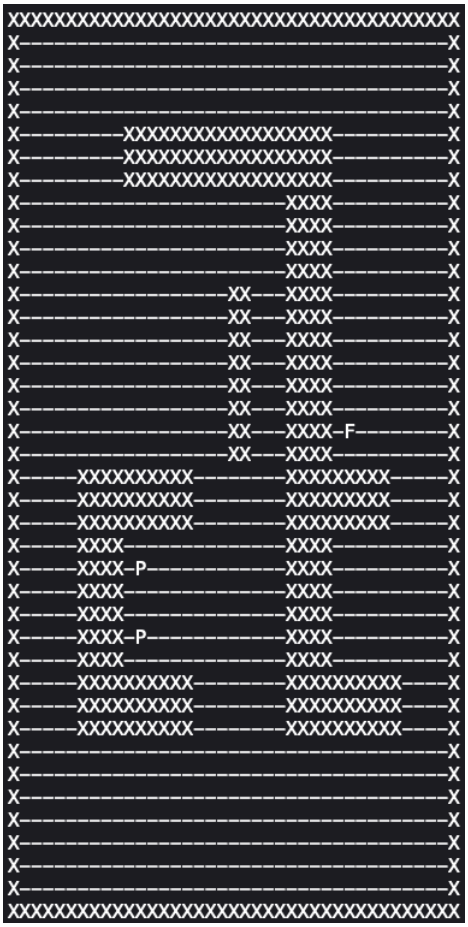



**Figura 19.** Circuito 11

Em caso de colisão vamos optar por dar vantagem ao menor custo até ao momento.


**Figura 20.** DFS VS BFS

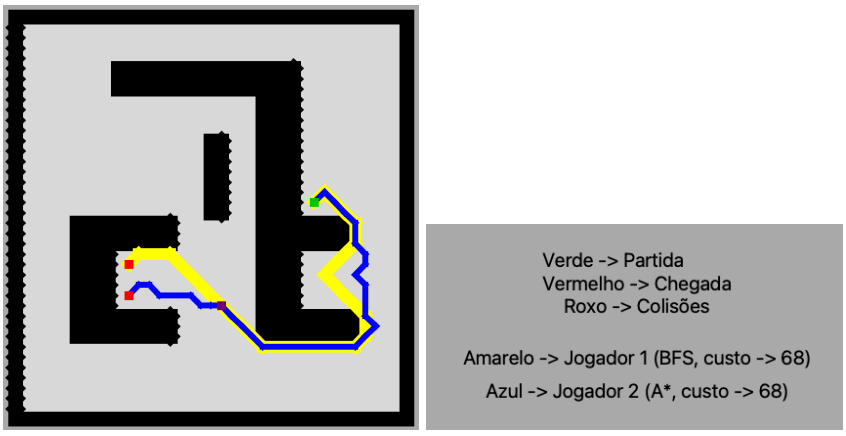



**Figura 21.** BFS VS A*

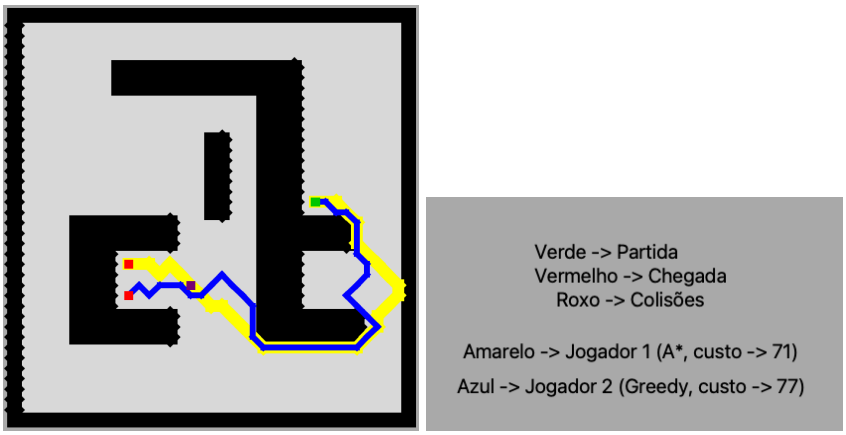



**Figura 22.** A* VS Greedy

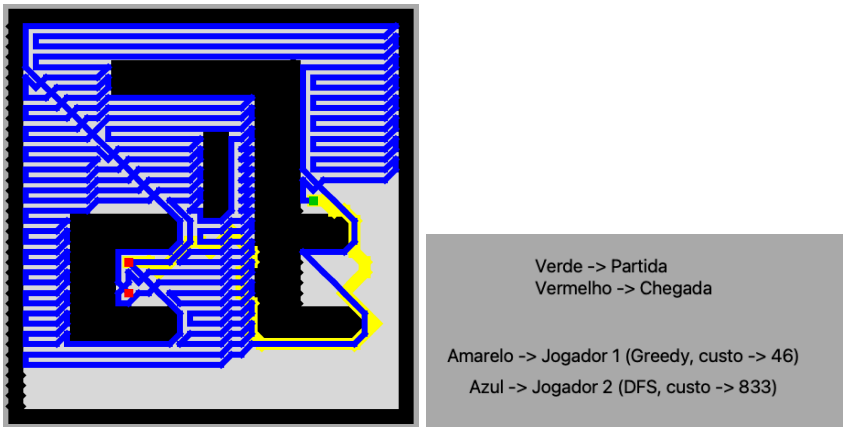



**Figura 23.** Greedy VS DFS

Na tabela seguinte é possíver verificar para cada Algoritmo o resultado obtido, o seu custo e, por fim, a existência de colisões.

| — Algoritmo 1 — | — Algoritmo 2 — | — Custo 1 — | — Custo 2 — | — Colisões — |
|---|---|---|---|---|
| DFS | BFS | 831 | 38 | 0 |
| BFS | A* | 68 | 68 | 1 |
| A* | Greedy | 71 | 77 | 1 |
| Greedy | DFS | 46 | 833 | 0 |

# 8 Extras

## 8.1 CheckPoints

O grupo teve a ideia de acrescentar também circuitos com *checkpoints*. Esta característica é particularmente útil para mapas circulares e obriga a que um jogador passe pelo checkpoint, não permitindo que este possa ir do ponto de partida para a meta diretamente, sem antes passar pelo checkpoint. Porém, devido ao curto tempo disponível por parte dos elementos do grupo, apenas implementamos esta estratégia para os Algoritmos A* e BFS e apenas fora do contexto multijogador.

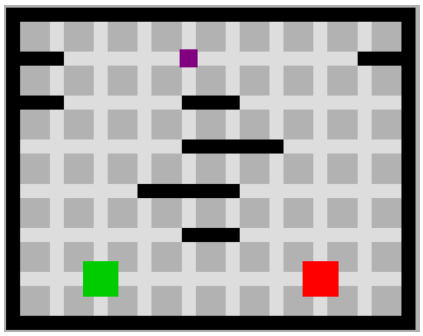


**Figura 24.** Exemplo Circuito com *CheckPoint*

```
    0  1  2  3  4  5  6  7  8  9
--------------------------------
0 | X  X  X  X  X  X  X  X  X  X
1 | X  X  -  -  7  6  -  -  X  X
2 | X  X  -  8  X  X  5  -  -  X
3 | X  -  9  -  X  X  X  4  -  X
4 | X 10  -  X  X  X  3  -  -  X
5 | X 11  -  -  X  X  2  -  -  X
6 | X  - 12  -  -  -  -  1  -  X
7 | X  X  X  X  X  X  X  X  X  X
```

**Figura 25.** Circuito c/ Checkpoint - BFS

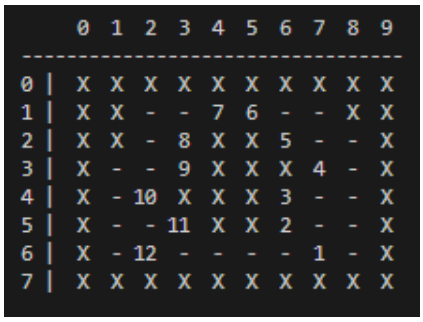



**Figura 26.** Circuito c/ Checkpoint - A*

# 9 Conclusão

## 9.1 Resultados Obtidos

Após várias execuções com Algoritmos e Circuitos diferenciados, foram retiradas as seguintes conclusões:

- Os algoritmos de procura não informada são muito pouco eficientes e não estão dentro dos moldes atuais de software inteligente, pois como já explicado, baseiam-se numa procura às cegas. É notável a diferença, apesar de, em alguns casos, devido também ao circuito, a BFS ficar bastante próxima do que seria o ideal.
- Os algoritmos de procura informada, por sua vez, são bastante mais competentes. Como era de esperar, os resultados obtidos dão vantagem a estes algoritmos, o que nos leva a acreditar que fomos bem sucedidos.

## 9.2 Comentário Final

Finalizando, sentimos que este projeto explorou a nossa criatividade e capacidade de superar dificuldades, o que nos levou a descobrir aspetos da Linguagem *Python* com os quais até então não estávamos familiarizados.

Apesar das adversidades encontradas ao longo da realização do projeto, conseguimos resultados bastante satisfatórios. A nível geral, e tendo em conta o que foi explicado nos capítulos anteriores, podemos afirmar que temos um projeto bem conseguido.

# Referências Bibliográficas


1. Draw.io: Ferramenta para construção gráfica da Topologia
2. StackOverflow: Ferramenta para resolver problemas relacionados com código e erros associados
3. Tkinter: Documentação consultada como auxílio na construção da parte gráfica
4. OverLeaf: Plataforma de colaboração em tempo real para a criação e edição de documentos $LaTeX$
5. Red Blob Games: Introduction to the A* Algorithm
6. Github: darioWorknet/A$_S tart - - Pygame$