Ball Sort Puzzle
Artificial Intelligence

André Malheiro up201706280@fe.up.pt

Diogo Gomes up201806572@fe.up.pt

Rúben Almeida up201704618@fe.up.pt

FEUP MIEIC IART Grupo43 2020/2021

Turma 3MIEIC07

Especificação do Problema e Referências Bibliográficas

Especificação do Problema:

Ball Sort Puzzle é um jogo do tipo puzzle cujo objetivo é alcançar uma ordenação homogénea das cores das bolas em todos o tubos que compõem o problema.

Ball Sort Puzzle originalmente é um jogo sem condição de vitória definida. Existe uma geração de níveis cada vez mais difíceis aos jogadores. Para efeitos de simplificação este pressuposto foi alterado. Passando a existir a consideração de vitória ao fim de um número finito de níveis ultrapassados

Implementações já existentes:

Existem várias produções deste jogo comerciais ou não. A produção sugerida como ponto de referência está disponível para Android na **PlayStore**.

Para além desta produção comercial que não disponibiliza o código fonte publicamente, para tópicos diferentes da GUI, relacionados com controlo e IA foram usado como referência múltiplos repositórios de Github não comerciais dos quais se destaca o **seguinte**, por fazer uso de heurísticas relevantes e do algoritmo A*

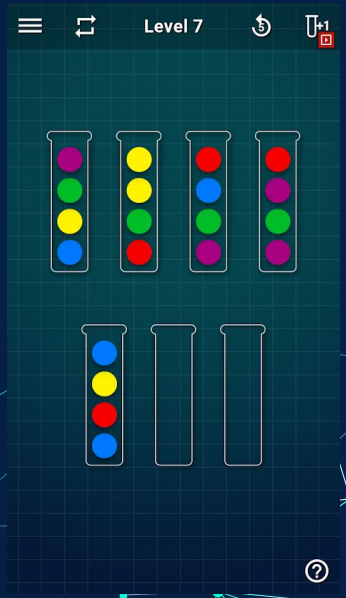


Fig.1:Layout da implementação em Android

Formulação do Problema

- **Representação do estado**

  Uma lista de tubos de capacidade N. Cada tubo por sua vez possui a lista das bolas que é portador.
  Cada bola é um objeto complexo que para além das "sprites" necessárias para as funções de rendering, mantém um Pair ternário representativo da sua cor em RGB.

  Implicitamente o estado pode ser abstraído como uma matriz de bolas.

- **Estado inicial**

  A lista de tubos é preenchida de acordo com os valores definidos para representar o nível 1. Um preenchimento com bolas de cor e quantidade aleatória.

  Poderão existir tubos que inicialmente não têm preenchimento

- Espaço de resultados

  De forma naïve podemos afirmar que os espaço de resultados possível para este problema é a quantidade de formas de colocar N bolas em Y tubos de forma indistinta.
  No entanto, dadas as restrições este valor raramente é atingido na prática.

  State Space(Ball Sort Puzzle) <= C (N_bolas, N_tubos)

- **Estado objetivo**

  Cada um dos tubos estará vazio ou terá de possuir uma lista homogênea de precisamente N bolas da mesma cor

- **Função de Avaliação / Heurísticas**

  No caso do Algoritmo A*, a formulação heurística é feita de acordo com a fórmula f(n)=g(n)+h(n), em que o custo da jogada [f(n)] é obtido pelo somatório de g(n), o custo real da jogada, e h(n) o valor da heurística.

  g(n) é avaliado com a simples quantificação do número de movimentos já executados. Os diferentes h(n) considerados são os seguintes:

  h1 - Quantificação da Entropia - Somatório do número de bolas que estão no local errado, ou seja, que perturbam a homogeneidade da cor da sequência de bolas de cada tubo.

  h2 - Quantificação da Distância à Homogeneidade - Somatório do número de movimentos que eram necessários para permitir a cada bola atingir o tubo cuja bola na base é da mesma cor sem contemplar regras.

  h3 - Inspirada em Tabu Search - Número de tubos não vazios + Somatório do número de bolas que estão no local errado * 2.
  Nota: Não produz solução ótima no A Star por não ser uma heurística optimista

Operadores

A implementação de Ball Sort Puzzle contempla somente o operador **Move**

Nome: **Move(Index_tubo_origem,Index_tubo_destino)**

- Index_tubo_origem- Índice no array de tubos do estado que corresponde o tubo de origem
- Index_tubo_destino- Índice no array de tubos do estado que corresponde o tubo de destino

Pré Condições:

- Não é possível retirar bolas a um tubo de origem que já esteja resolvido (preenchido com 4 bolas homogéneas)
- Não é possível retirar bolas a um tubo de origem vazio [len(balls)=0]
- Não é possível inserir bolas a um tubo de origem que esteja no máximo da sua capacidade [len(balls)=capacity(tube)]
- **Se as condições anteriores se verificarem:**

-É possível movimentar as bolas entre tubos que a cor da bola no topo seja igual

ou

-Quando o destino é um tubo vazio

Efeitos:

- O tubo de destino recebe a bola que é retirada na origem.
  [tubes[index_tubo_destino].append(tubes[index_tubo_origem].pop()]

Custo: Todas as operações têm custo unitário de 1

Detalhes de Implementação

- Linguagem de programação - Python 3.8
- Arquitetura adaptada de MVC usando paradigma OOP
- Bibliotecas usadas - GUI: Pygame 2
- Ambiente de Desenvolvimento: Pycharm
- Estruturas de Dados:

  - Listas de Tubos - Listas de Bolas -Queue para visitar os nós -Queue para guardar nós visitados

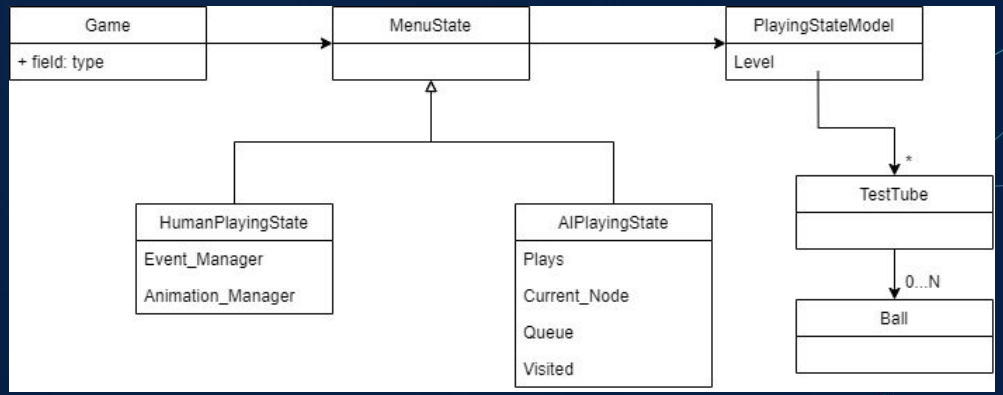


Fig.2:Esquema UML da estrutura central de Controlo do programa

Algoritmos e GUI Implementadas

No que respeita aos algoritmos de pesquisa, foram implementados:

-BFS -DFS -Iterative Deepening - Greedy -A Star

Dada a semelhança na implementação de BFS/DFS/Greedy/A Star, e tirando partido da abordagem OOP, fizemos uso do ***template pattern***.

Este padrão de design permite construir famílias de algoritmos que possuem um tronco comum, mas que certas etapas são executadas de forma diferente dependendo do elemento da família com que queremos trabalhar.
Esta abordagem simplificou a forma como se colocam/extraem os nós da fila auxiliar de pesquisa no loop do jogo.

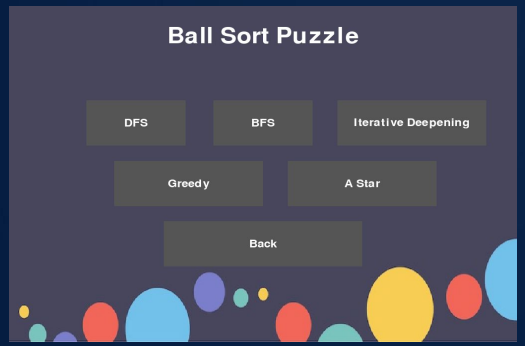


Fig.3: Modos de Jogo possíveis para Bot

No que respeita à GUI, foi implementada:

De forma de criar uma UX com a melhor relação custo-benefício, criamos uma GUI com capacidade de reproduzir som usando a biblioteca de *Python* **Pygame 2**

O modo primordial de interação com a GUI é usando a tecla esquerda do rato.

No modo de *Bot* não existe qualquer processamento de input durante o jogo.

Durante o modo de humano, podemos selecionar os movimentos na forma origem->destino usando o rato. Estes movimentos são acompanhados quer de animações sonoras, quer gráficas para potenciar a experiência do utilizador.

Para requerer hints, podemos clicar, no modo humano, na tecla "h". Será mostrado no ecrã a sugestão da AI para aquele momento.

Por último, durante o jogo foram introduzidos *headers* relevantes para dar render de estatísticas em real time do que está a acontecer no *background*

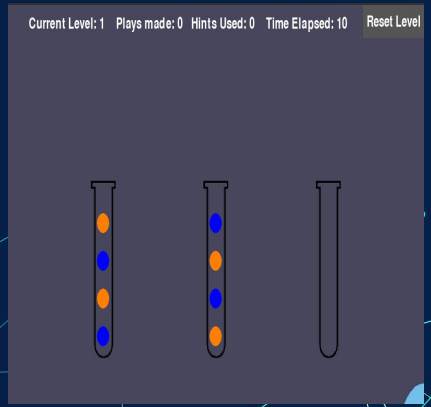


Fig.4: Imagem do layout atual da solução

Heurísticas Implementadas

Utilização de heurísticas de forma a executar pesquisa orientada

-H1 que quantifica a entropia [3],

-H2 que quantifica a distância mais próxima à homogeneidade, [3],

-H3 um heurística que penaliza com fatores multiplicativos os estados altamente entrópicos que queremos evitar. [3],

- Uma análise teóricas *à priori* às três heurísticas permite afirmar que h3 por ser a única heurística não optimista, não irá produzir um resultado ótimo de solução na aplicação do algoritmo A Star.

- Tal como demonstrado nos gráficos a seguir, esse pressuposto manteve-se o que nos permitiu adquirir um grau elevado de confiança na solução encontrada.

- Contudo, para além deste pressuposto, a não obtenção de uma solução ótima **nada permite à priori discernir da qualidade das heurística**, no que diz respeito à performance temporal e espacial na busca de solução. Essa análise efetua-se de forma empírica na secção deste documento dedicada à apresentação de gráficos comparativos.

Utilização de heurísticas para deteção de final de jogo e produção de hints

-Ball Sort Puzzle pode terminar, nomeadamente em níveis difíceis, **em estados impossíveis cíclicos.**

-Os níveis avançados de Ball Sort Puzzle são construídos para induzir o jogador em erro. **As melhores jogadas locais tendem a conduzir a derrotas.**

Da impossibilidade temporal de executar um A Star a cada interação para deteção de derrota e a cada pedido de uma hint, **introduzimos duas heurísticas que definem o nível máximo de *lookahead*** para estas passagens intermédias.

-Lookahead para deteção de derrota: - Profundidade de 8
-Lookahead para produzir melhor hint: - Profundidade de 15

Gráficos Performance Temporal/Espacial

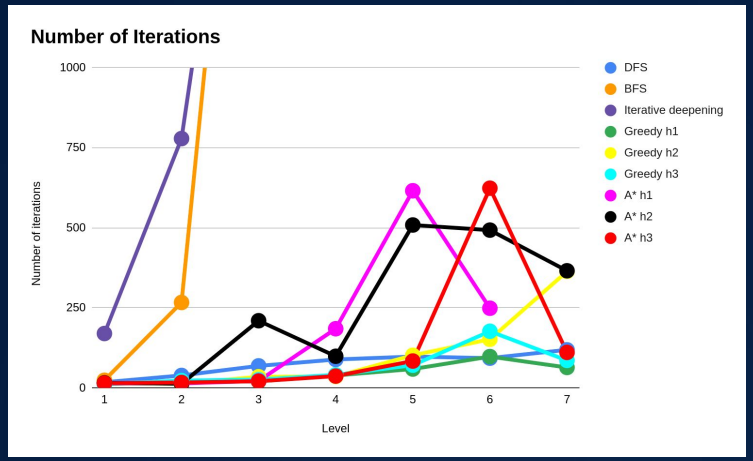


Gráfico.1: Gráfico de comparação do número de iterações (proporcional ao tempo) da execução dos algoritmos

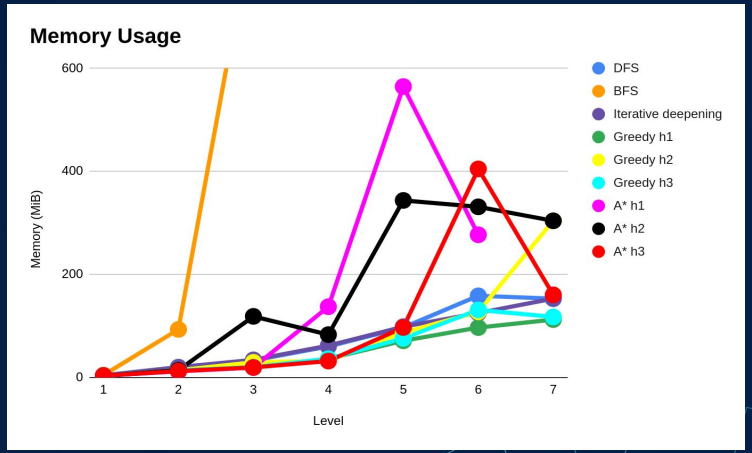


Gráfico 2: Gráfico de comparação da memória utilizada na execução dos algoritmos

Os gráficos acima observados demonstram a performance dos algoritmos implementados a nível temporal como a nível espacial.

Os resultados obtidos encontram-se todos dentro do esperado tendo em conta a complexidade temporal e espacial de cada algoritmo, podendo-se salientar o facto das diferentes heurísticas, utilizando o algoritmo A Star, terem diferentes performances de acordo com o nível onde estas estão a executar. Isto deve-se às formas distintas como cada uma delas aborda o respetivo cálculo onde uma estimativa pode levar a uma aproximação à solução mais rapidamente e com menos uso da memória do que a outra.

Gráfico Qualidade da Solução

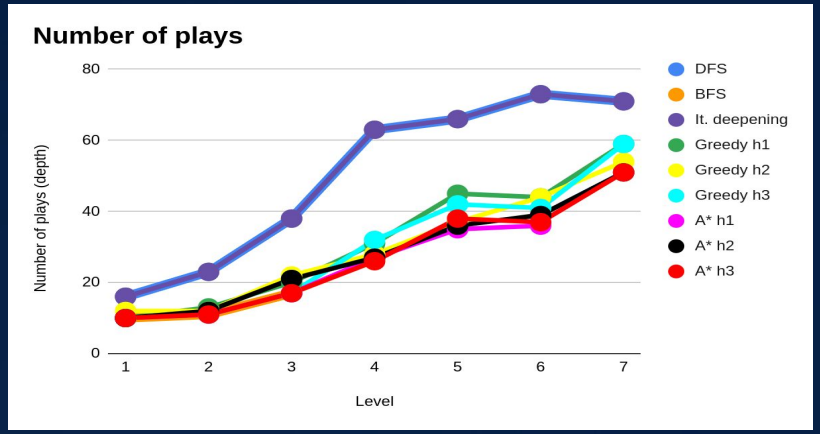


Gráfico.3: Gráfico de comparação do número de jogadas da execução dos algoritmos

Os gráficos acima observado demonstra o número de jogadas que as soluções de cada algoritmo proporcionam.

Analisando este gráfico, como era de esperar o algoritmo A Star apresenta as soluções mais próximas do ótimo de cada nível com pequenas variâncias de heurística para heurística originadas pelas razões explicadas no slide anterior.

Utilizando a heurística 2 foi observado um resultado fora do esperado (até acima do greedy com a mesma heurística) talvez por uma má estimação da mesma heurística. O mesmo não se observa nos diferentes níveis pelo que podemos concluir que o erro se baseia numa situação em particular.

Conclusões

Através do desenvolvimento de Ball Sort Puzzle foi possível aprofundar o conhecimento da temática de resoluções por pesquisa de problemas de IA.

Ficou claro a importância da utilização de heurísticas para conseguir guiar o algoritmo no caminho correto. De forma análoga ficou também claro as dificuldades que problemas deste tipo acarreta, sobretudo dada ao rápido crescimento dos espaços de estados

O desenvolvimento conseguiu alcançar os objetivos exigidos, foram implementados todos os algoritmos com os resultados previstos. Também os dois modos de jogo previstos, humano e bot, foram implementados. Por fim, foi possível chegar a uma solução satisfatória para a geração de hints ao jogador.

Para além disso foi extrapolado a exigência exigida quanto às questões de UX e GUI com a utilização do pygame para obtenção de resultados muito satisfatórios neste sentido.

Como trabalho futuro, os autores sugerem a implementação de um caso de uso extra também ele relacionado com IA, a capacidade de geração de níveis com diferentes graus de dificuldade de forma automática.

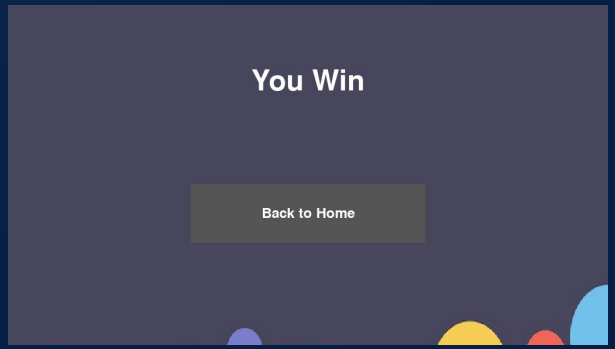
Referências Bibliográficas

Página de Ball Sort Puzzle na PlayStore: https://play.google.com/store/apps/details?id=com.spicags.ballsort&hl=pt_PT&gl=US Consultado em : 08/03/2021

Página de Resolução de nível de Ball Sort Puzzle: https://levelsolved.com/ball-sort-puzzle/ Consultado em : 29/03/2021

Lottem, Ran. Ball-Sorter public github repository: https://github.com/Krumpet/ball-sorter Consultado em : 10/03/2021

Reis, Luís Paulo, Lecture 2b: Solving Search Problems.



Anexo A2.1-Tabela relativas aos gráficos de análise temporal

Apresentação das tabelas com os dados anotados que serviram de base aos gráficos apresentados nos primeiros slides deste documento

ITERATIONS - PROF 5 5

Level	DFS	BFS	Iterative deepening	Greedy h1	Greedy h2	Greedy h3	A* h1	A* h2	A* h3
1	18	24	170	12	15	12	15	16	15
2	39	267	779	18	13	23	14	13	17
3	69	2695	2414	21	35	26	22	210	21
4	89		4004	39	36	41	185	99	37
5	98		4850	59	102	71	616	509	84
6	93		4370	98	152	177	249	493	624
7	119		7139	64	364	86		366	111

TIME ELAPSED

Level	DFS	BFS	Iterative deepening	Greedy h1	Greedy h2	Greedy h3	A* h1	A* h2	A* h3
1	0,131	0,194	0,913	0,078	0,097	0,07	0,11	0,099	0,096
2	0,795	8,771	9,305	0,273	0,216	0,297	0,213	0,228	0,245
3	1,67	285,33	34,375	0,386	0,689	0,519	0,396	6,016	0,375
4	2,738		75,909	0,883	0,609	0,764	4,646	2,057	0,717
5	3,554		117,815	1,617	2,411	1,861	40,215	21,598	2,37
6	3,494		114,378	2,1	4,148	4,557	12,761	18,374	27,429
7	4,579		196,38	2,679	12,404	2,91		12,562	4,537

Tabela 1: Tabela referente ao gráfico 1

Anexo A2.2-Tabela relativas aos gráficos de análise espacial

Level	DFS	BFS	Iterative deepening	Greedy h1	Greedy h2	Greedy h3	A* h1	A* h2	A* h3
1	4,3	5	4,4	3,20	3,7	3,1	3,8	3,7	3,8
2	19,8	93,6	20,4	13,9	14,40	14,2	12,2	14,4	12,9
3	33,5	886,6	34,5	19,8	29,7	20,4	19,7	118,9	19,8
4	60,5		62,1	36,4	32,9	36,8	137,6	83,5	32,2
5	97,2		98,9	71,5	87,4	76,2	564,5	343,7	97,3
6	158,7		125,4	97,2	127,7	131,4	277,1	331,3	404,8
7	153,3		153,3	112,6	304,3	117,8		304,2	160,6

Tabela 2: Tabela referente ao gráfico 2

Anexo A2.3-Tabela relativas aos gráficos do número de passos da solução

Number of plays (PROF (5,5))									
Level	DFS	BFS	Iterative deepeni	Greedy h1	Greedy h2	Greedy h3	A* h1	A* h2	A* h3
1	16	10	16	10	12	10	10	10	10
2	23	11	23	13	12	11	11	12	11
3	38	17	38	20	22	17	17	21	17
4	63		63	31	28	32	27	27	26
5	66		66	45	37	42	35	36	38
6	73		73	44	44	41	36	39	37
7	71		71	59	54	59		51	51

Tabela 3: Tabela referente ao gráfico 3

Anexo A3.1-Análise da performance temporal em função da profundidade a que são testados duplicados

Apresentação de uma análise inconclusiva à variação do nível de profundidade relativa entre nós em que testamos a existência de duplicados.
Os autores não foram capazes de extrair resultados relevantes que fossem relevantes de inserir nas primeiras páginas desta apresentação

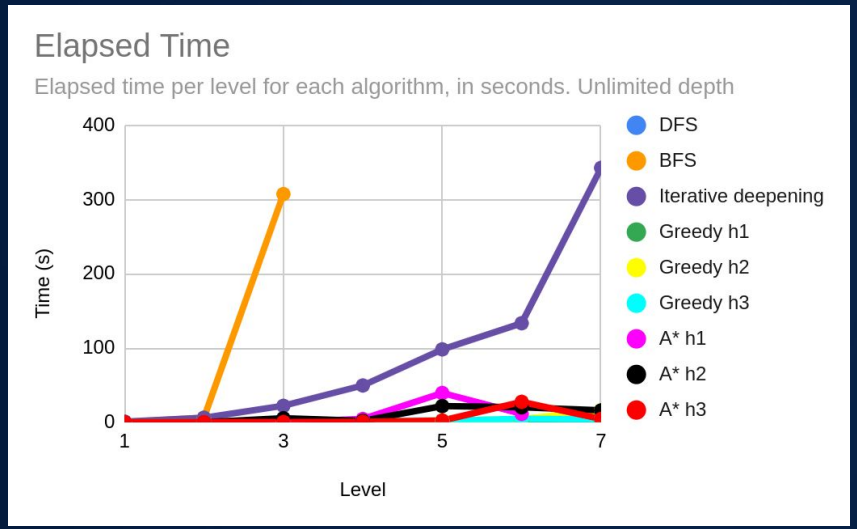


Gráfico 4: Teste com profundidade ilimitada

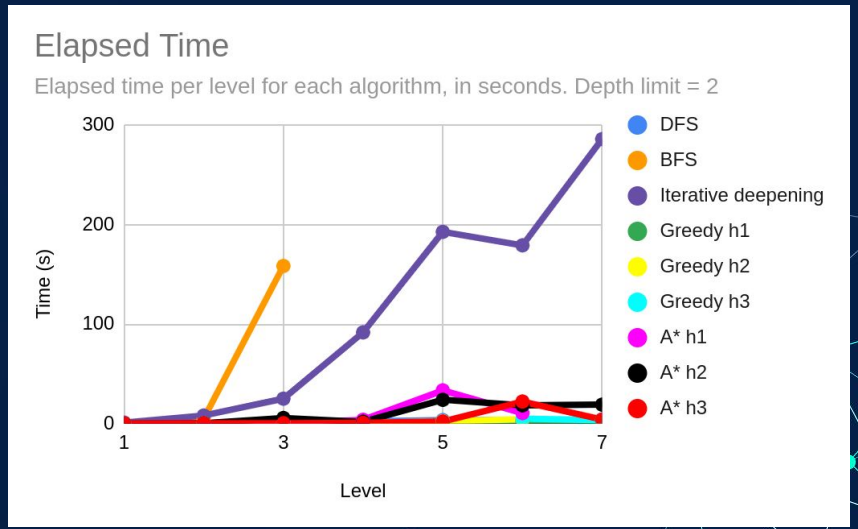


Gráfico 5: Teste com profundidade 2

Anexo A3.2-Análise da performance temporal em função da profundidade a que são testados duplicados

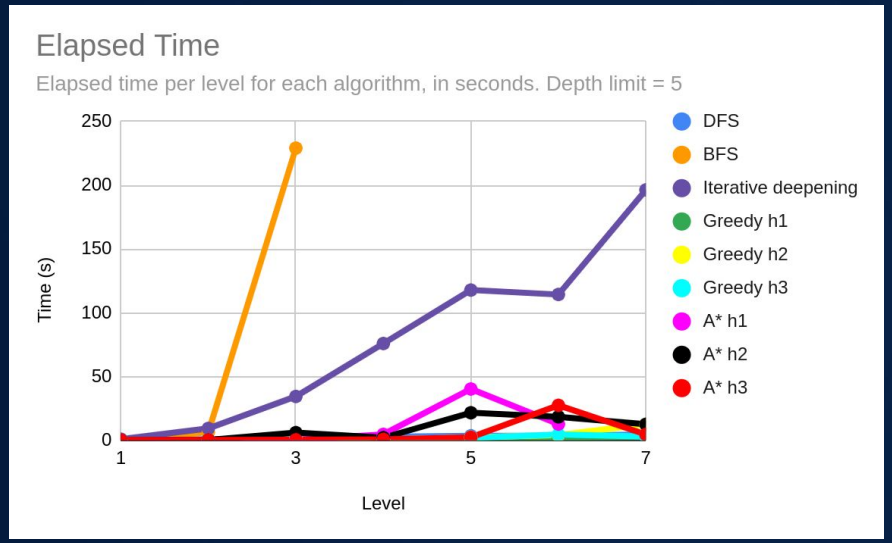


Gráfico 6: Teste com profundidade 5

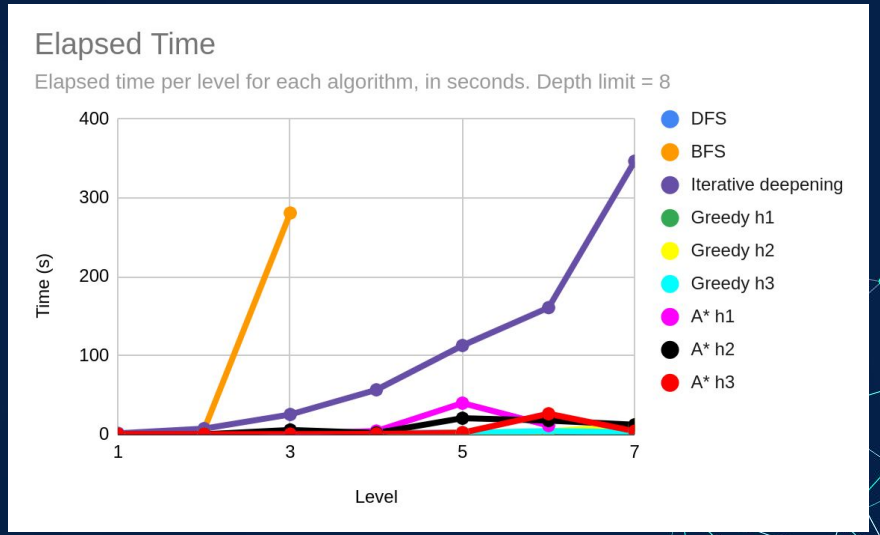


Gráfico 7: Teste com profundidade 8