Inteligência Artificial

Mestrado Integrado em Engenharia Informática e Computação

3º Ano - 2º Semestre

Pesquisa Aplicada à Resolução do Jogo Hopeless

Inês Santos Carneiro - up201303501@fe.up.pt

João Oliveira e Silva - up201305892@fe.up.pt

Pedro Vieira de Castro - up201305337@fe.up.pt

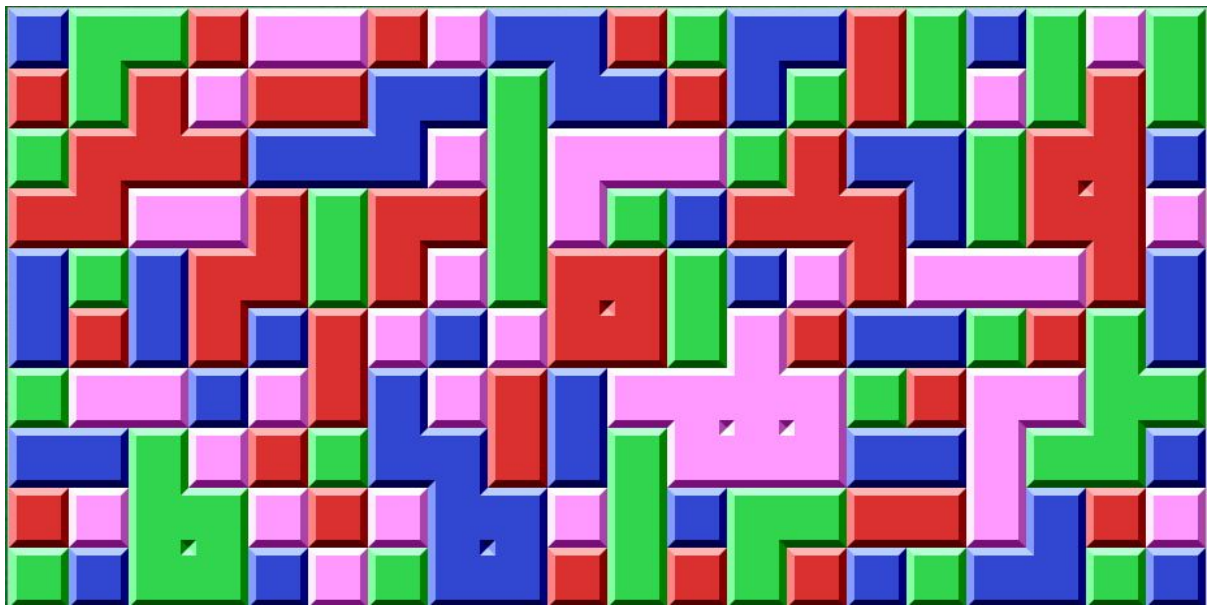
Objetivo

O projecto desenvolvido é capaz de obter as jogadas necessárias para maximizar a pontuação num tabuleiro do jogo Hopeless.

Estas jogadas irão ser calculadas a partir de um dos vários algoritmos de pesquisa que foram estudados nesta Unidade Curricular (nomeadamente **A***).

Vai ser possível comparar cada um dos algoritmos, quer em termos de tempo de processamento, uso de memória e pontuação final, permitindo assim avaliar qual o melhor a ser usado.

Para facilitar a interpretação destes dados, desenvolvemos uma interface gráfica que vai permitir jogar o jogo e mostrar as jogadas que os diferentes algoritmos obtiveram.



Especificação

O Jogo

O objectivo do Hopeless é obter o maior numero de pontos eliminando blocos do tabuleiro. Um bloco é um conjunto de quadrados adjacentes da mesma cor, que criam uma região. Os pontos que um bloco vale depende da quantidade de quadrados que esse bloco tem.

$$Pontos = (2(n-1))^2$$, em que n = nº de quadrados

Após a eliminação dum bloco, todos os quadrados que tenham sob eles um espaço vazio irão “cair”, de forma semelhante ao Tetris.

No caso de uma coluna se encontrar completamente vazia, as colunas á direita serão deslocadas para a esquerda ocupando assim a coluna em falta.

O jogo acaba quando não há mais blocos de dimensão 2 ou superior, isto é quando já não existem mais jogadas possíveis. Este estado vai ser referido como estado final.

Os tabuleiros iniciais serão gerados de forma aleatória com o seu tamanho e o nível de dificuldade (nº de cores diferentes) como parâmetros variáveis.

Algoritmos Implementados

Nesta secção vão ser apresentados os algoritmos implementados tanto ao nível do seu funcionamento como da qualidade dos resultados obtidos. A solução será um conjunto de jogadas que podem ser aplicadas por ordem ao tabuleiro original.

Depth-First Search (DFS)

O **Depth-First Search** (em português Pesquisa em Profundidade) é um algoritmo desenhado para percorrer árvores/grafos. Este algoritmo progride expandindo o primeiro nó arbitrário não explorado encontrado e assim se aprofundar até que chegue a uma solução ou encontrar um nó que não possua filhos/vizinhos por explorar, sendo que nesse momento ocorre *backtracking*.

No âmbito deste trabalho, este algoritmo não é considerado útil visto que os resultados obtidos não possuem qualquer qualidade devido ao facto de qualquer nó levar a um estado final e portanto não ocorrer qualquer *backtracking*.

Breadth-First Search(BFS)

O **Breadth-First Search** (em português Pesquisa em Largura) é um algoritmo desenhado para percorrer árvores/grafos. Este algoritmo progride expandindo todos os nós filho/vizinho não explorados. Para cada um destes será feito o mesmo até que seja encontrada uma solução. O algoritmo termina assim que um nó expandido seja uma solução.

No âmbito deste trabalho, a solução obtida por este algoritmo apresenta um (ou o único) conjunto de menor jogadas possíveis para chegar ao estado final. Devido à natureza do jogo, o menor numero de jogadas não reflete necessariamente uma solução ótima.

Na implementação deste algoritmo há uma opção que nos permite expandir todos os nós. Através deste método de **Brute Force** é nos possível achar sempre o valor máximo da pontuação para o jogo. No entanto devido ao grande poder

computacional e memória necessária para tabuleiros de grande dimensão é impensável usar qualquer um destes métodos.

Iterative Deepening Depth-First Search (IDDFS)

O **Iterative Deepening Depth-First Search** (em português Pesquisa em Profundidade com Aprofundamento Iterativo) é um algoritmo de pesquisa de solução baseado no **Depth-First Search**. Este algoritmo progride da mesma forma que o seu predecessor com a diferença que lhe é limitado o nível de aprofundamento que quando é atingido é forçado a ocorrer *backtracking.* De iteração em iteração este nível é aumentado.

Este algoritmo é equivalente ao **Breadth-First Search.** A diferença entre ambos é que este troca a recursos de memória por tempo computacional.

Greedy Search

O **Greedy Search** (em português Pesquisa Gulosa) é um algoritmo de pesquisa baseado **Depth-First Search**. Este algoritmo progride da mesma forma que o seu predecessor com a diferença que a escolha do nó a ser expandido não é arbitrária. Aqui será escolhido aquele nó cuja pontuação da jogada seja superior.

No âmbito deste trabalho, este algoritmo não é considerado útil. Devido à natureza polinomial da pontuação do jogo por vezes é necessário fazer uma "má" jogada, isto é, uma jogada de pontuação baixa para de seguida compensar com uma jogada melhor.

MaxMin

O algoritmo MaxMin é um algoritmo inventado por nós, recorrendo a uma pesquisa heurística.

Foi possível criar este algoritmo devido a uma propriedade dos tabuleiros Hopeless. A propriedade permite que seja possível calcular o minimo de pontos possíveis de fazer com um tabuleiro. Isto acontece porque todas as jogadas possíveis de fazer num dado tabuleiro podem ser executadas. Isto permite atribuir

um minimo de pontuação a cada tabuleiro e o valor heurístico que servirá para comparar a promissoriedade de cada tabuleiro.

A partir disto, o algoritmo MaxMin expande a jogada cujo tabuleiro resultante permite obter o máximo das minimas pontuações. O algoritmo termina quando atingir um estado final.

A necessidade de desenvolver este algoritmo partiu da necessidade de criar um algoritmo que escalasse de forma linear com o numero de jogadas e ao mesmo tempo que mantivesse uma boa qualidade de pontuação final. Por este motivo este é o algoritmo recomendado para tabuleiros de grande dimensão.

A*

O **A*** é um algoritmo de pesquisa informada. **A*** é similar ao algoritmo de **Dijkstra** na medida em que tem como objectivo achar o caminho de **menor** custo até à solução.

O **A*** progride expandido o nó mais promissor até achar uma solução. O nó mais promissor é aquele que cujo valor de f*(n) é menor. Os filhos/vizinhos deste nó vão ser avaliados e inseridos numa lista de prioridade denominada por *OPEN*. O algoritmo termina a sua pesquisa assim que for encontrada uma solução, assim que esteja num estado final h*(n) = 0 e f*(n) = g(n).

A avaliação será feito a partir da seguinte função:

$$f*(n) = g(n) + h*(n)$$

Em que n é o nó a ser avaliado, g(n) o custo para chegar ao nó n e h*(n) será o valor heurístico do custo estimado para chegar à solução.

Para o algoritmo oferecer a solução ótima a heurística utilizada tem que ser **admissível**.

Diz-se uma heurística **admissível** se o valor do custo até à solução não for sobrestimado.

Aplicação de A* ao Hopeless

Anteriormente foi descrito a forma como o algoritmo **A*** se comporta. No entanto, a aplicação do algoritmo ao jogo requer algumas modificações ao algoritmo original.

Minimização vs Maximização

A descrição diz que o **A*** "*tem como objectivo achar o caminho de **menor** custo*". No entanto, quando falamos em maximizar este não é o objectivo. No caso de maximização o nó mais promissor será aquele com valor de f*(n) superior a todos os outros.

Agora h*(n) vai estimar a recompensa a chegar ao estado final em vez do que anteriormente seria o custo.

Função Heurística

A definição da admissibilidade de uma heurística também terá que ser refeita por se tratar de um problema de maximização.

O ajuste necessário à definição é o seguinte:

O valor de h*(n) vai ser ser sempre superior ou igual à recompensa até ao estado final ótimo a partir de n.

Por outras palavras vai ter que ser sempre superior à pontuação que pode ser obtida no máximo a partir desse nó.

A única forma da admissibilidade ser assegurada é criar uma função que calcule a pontuação se todos os pontos que estão em jogo estivessem posicionados de forma ideal.

A equação da função heurística é a seguinte:

$$h*(n) = 0$$
$$foreach\ different\_color\ as\ color$$
$$\quad h*(n) \mathrel{+}= Points(numberOfSquares(color))$$

Esta pontuação vai sobrestimar constantemente. O problema está no facto de não ser possível aproximar este valor da pontuação real sem simular jogadas. Porém, é exatamente simular jogadas o objectivo do algoritmo.

Estado Final

Ao avaliar um estado final o valor de h*(n) pode resultar num valor superior a 0. Isto é resultado de uma possível sobrestimação do valor heurístico.

> *"Therefore, while the f(t) upper bounds the optimal solution, g(t) may not, since h(t) may be larger than 0. Thus, unlike **A*** for MIN problems, expanding a goal with **A*** in MAX problems does not guarantee that the optimal solution has been found."*

Como h*(n) é na prática a recompensa até ao estado final não faz sentido o valor ser diferente de 0. No entanto como é possível que g(n) não seja a ótima, então a solução encontrada pode não ser a ótima mesmo que h(n) = 0.

No entanto podemos usar h(n) = 0. Se assim o fizermos estaremos a fazer uma pesquisa por mais nós visto que ao estarmos a minimizar o valor de f*(n), consequentemente deixando que nós em *OPEN* sejam expandidos que com valor de h*(n) > 0 não seriam.

No trabalho final esta opção é deixada ao critério do utilizador.

Iterative Deepening A*

A implementação do algoritmo **Iterative Deepening A*** (em português A* com Aprofundamento Iterativo) não faz sentido quando estamos a falar de maximização. Definir um limite para f* não é beneficial visto que estamos à procura de o maximizar. Por esta razão o algoritmo não foi implementado.

Desenvolvimento

Linguagem de Programação/IDE/API's Utilizadas

O trabalho foi realizado na íntegra em Java. O IDE utilizado foi IntelliJ IDEA Community Edition.

A Graphical User Interface foi implementada com o auxilio da API Swing.

As estatísticas que são exportadas diretamente para Microsoft Excel. Usamos a Apache POI API para este propósito. Esta API permite a interação eficiente e acessível de Java com produtos Microsoft Office.

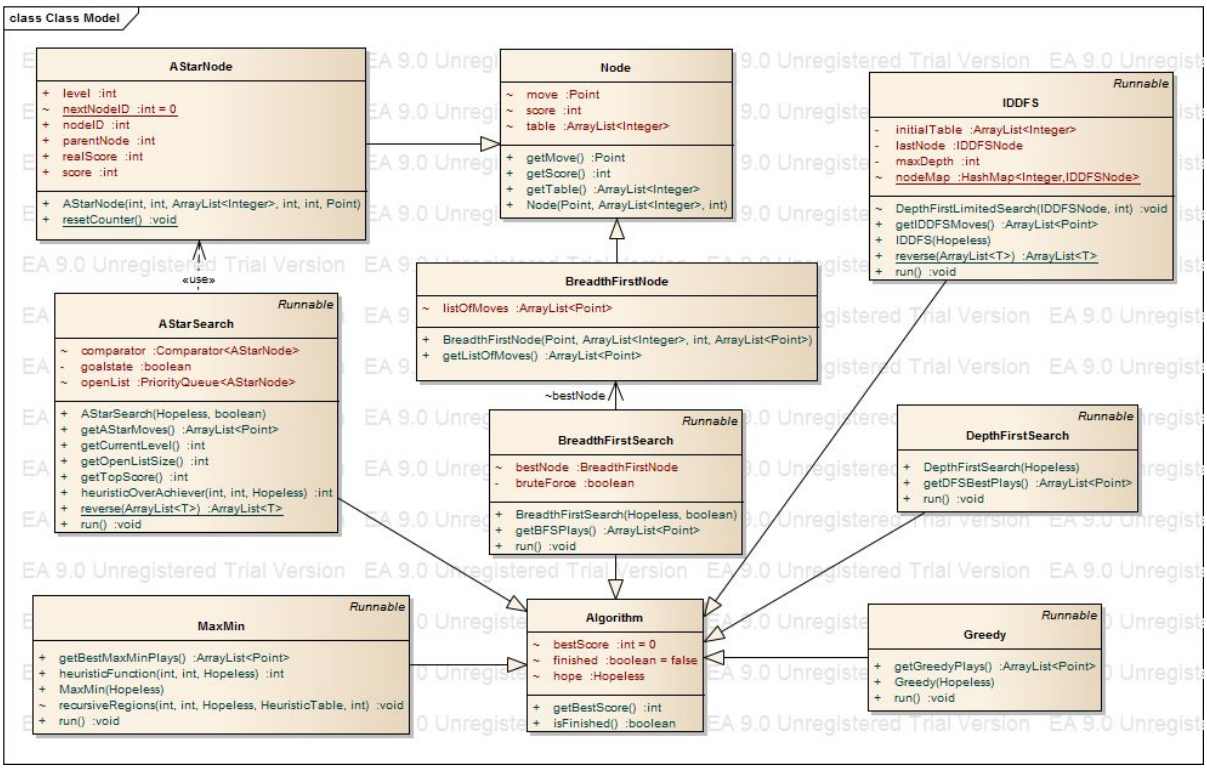
Estatísticas Geradas

Os vários algoritmos têm tempos de execução e obtêm resultados diferentes. Implementamos uma função que gera um certo numero de tabuleiros distintos (por user input). Por estes tabuleiros são corridos todos os algoritmos. São registadas as pontuações e o tempo de execução de cada algoritmo e é feita a média destes valores para cada algoritmo. A partir destes valores calculamos a eficiencia de cada um destes algoritmos (a falar mais à frente).

Como anteriormente foi mencionado, tudo isto é gerado automaticamente num ficheiro Microsoft Excel chamado "*statistics.xlsx*" assim que é chamada a função.

Arquitetura do Sistema

De seguida segue-se o *UML* de classes no estado final do projecto.



Todos os algoritmos estendem a classe Algoritmo que possui propriedades e funções comuns a todos os algoritmos.

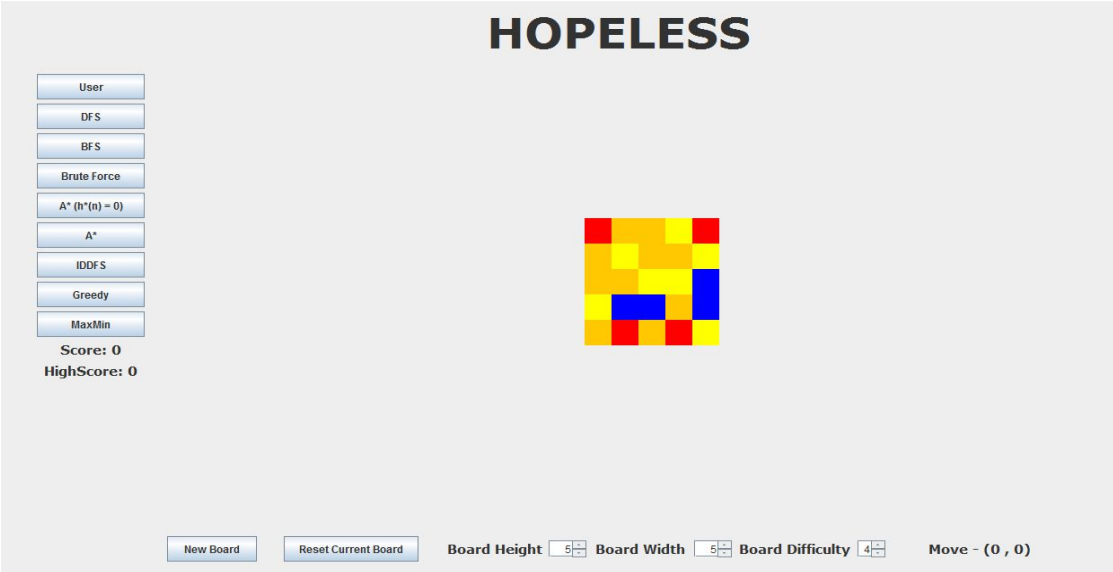
Alguns dos algoritmos têm a necessidade de registar os nós por onde já passaram, por este motivo criamos a classe Node.

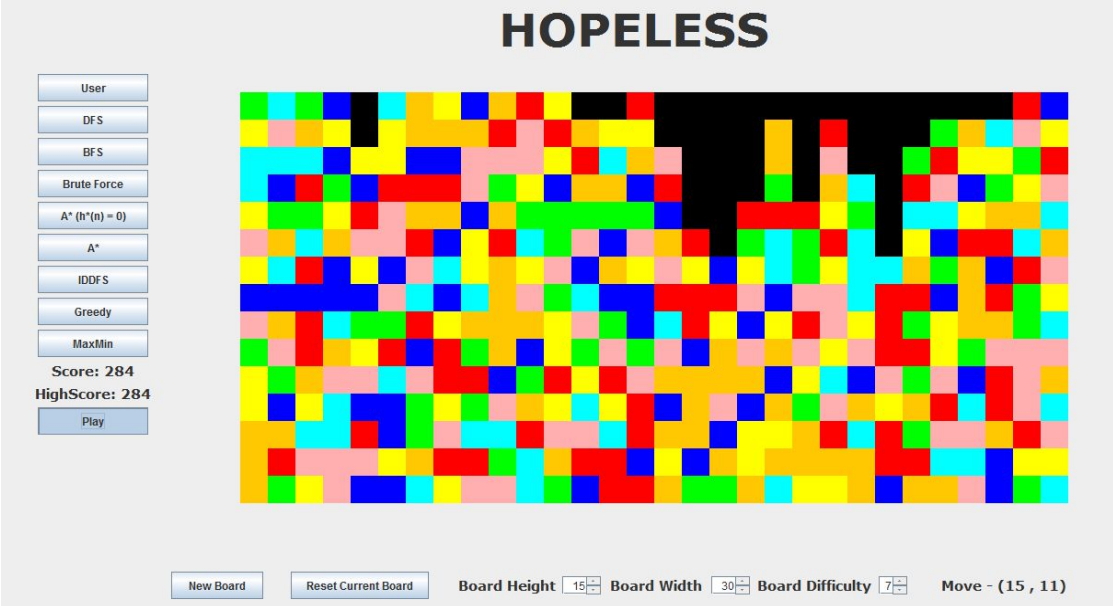
Graphical User Interface

Uma aplicação gráfica foi desenvolvida em Swing no âmbito deste projeto. Esta aplicação permite ao utilizador criar um tabuleiro com a dimensão e dificuldade que for desejada.

Para além disso permite ao utilizador accionar um de dois modos:

1. Jogar o jogo, o utilizador clica nos blocos com o rato.
2. Escolhe um dos algoritmos e é possível ver os passos calculados pelo algoritmo a serem executados.





Experiências

Graças à nossa função de estatísticas é nos possível gerar grande quantidade de tabuleiros aleatórios e sobre eles executar cada um dos algoritmos.

Devido à complexidade temporal e espacial de algum dos algoritmos não pudemos gerar este tipo de estatísticas com tabuleiros de grande dimensão visto que alguns tornam necessária a expansão de todas as hipóteses. Para um tabuleiro 20x10 (tamanho original) a árvore de pesquisa criada pode chegar aos $40! = 8.2 * 10^{47}$ (40 vem de uma média de 40 jogadas iniciais possiveis).

Por esta razão decidimos gerar tabuleiros de tamanho 5x5 com dificuldade de 4 (4 cores diferentes).

Geramos 5000 tabuleiros 5x5 com dificuldade de 4 porque achamos que seria uma amostra significativa dos possíveis tabuleiros iniciais.

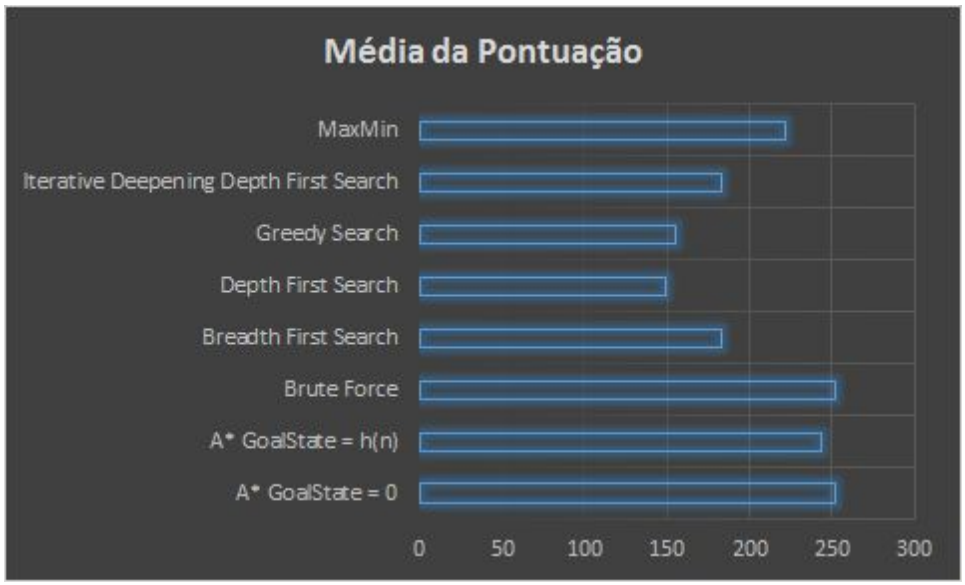
Resultados

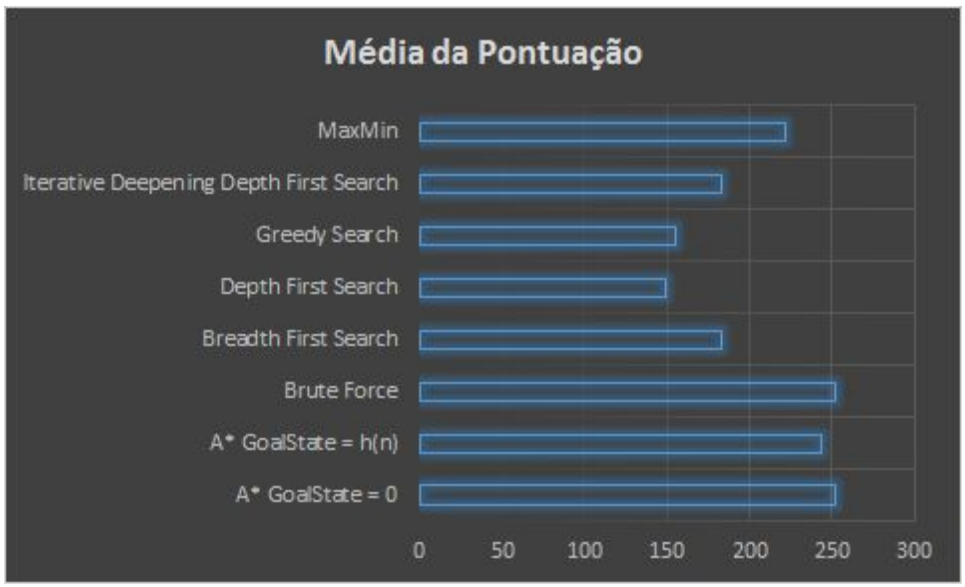
Um dos cálculos que fazemos é calcular a eficiência de cada algoritmo. Definamos a eficiência como a razão entre a média da pontuação com o tempo de pesquisa. Quanto melhor a eficiência melhor a relação de qualidade/tempo.

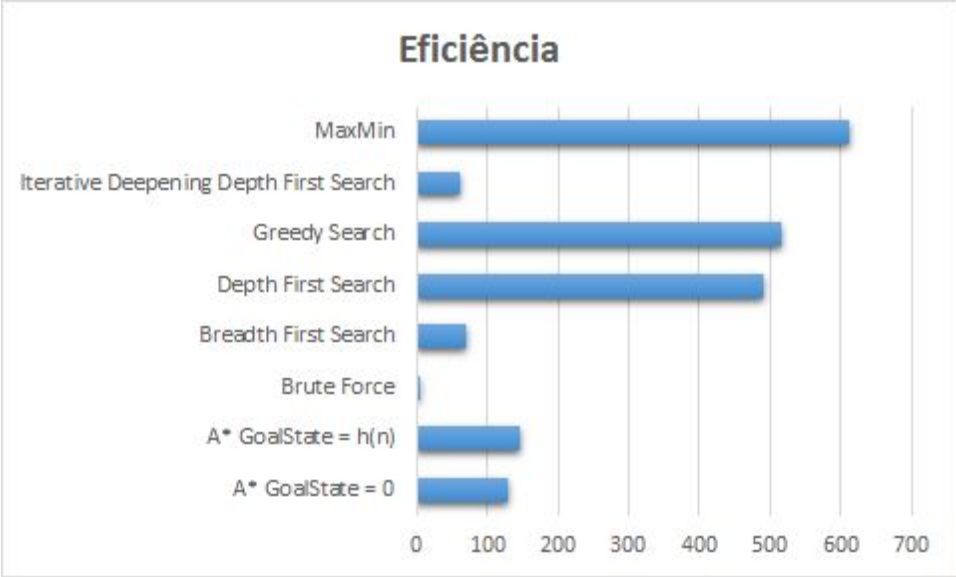
	A* GoalState = 0	A* GoalState = h(n)	Brute Force	Breadth First Search
Média de Pontuação	251,9328	243,864	251,9328	182,8622
Tempo de Execução	1,962263	1,656838	46,90228	2,648639
Eficiência	128,3889	147,1864	5,37144	69,04005

	Depth First Search	Greedy Search	Iterative Deepening Depth First Search	MaxMin
Média de Pontuação	148,7522	155,3735	182,8622	221,3491
Tempo de Execução	0,302649	0,301195	2,941633	0,362163
Eficiência	491,5	515,8569	62,1635	611,1855

Retiramos destes dados os seguintes gráficos:







Análise dos Resultados

Há bastantes conclusões interessantes que podem ser concluídas a partir dos dados retirados.

1. Os algoritmos **Breadth First Search** e **Iterative Deepening Depth First Search** resultam exatamente no mesmo resultado para cada um dos tabuleiros, como seria de esperar. Mas ainda é possível analisar uma propriedade do **Iterative Deepening Depth First Search.** É previsto que este algoritmo calcule **11%** mais nós que a sua contra-parte e isto é possível observar no tempo de execução, em que este possui um valor aproximadamente **11%** superior ao do **Breadth First Search.**
2. O **Depth First Search** é o algoritmo mais rápido. Isto provém do facto que o algoritmo não tem que calcular qualquer escolha e que qualquer jogada leva a um estado final válido. Desta forma não ocorre qualquer *backtracking*.
3. O algoritmo menos eficiente e mais lento é o **Brute Force**. Isto acontece porque é o único que passa por todos os nós da árvore de pesquisa.
4. O algoritmo mais eficiente é o **MaxMin**. Isto porque é uma combinação do **Depth First Search**, o algoritmo mais rápido, com uma avaliação heurística.

Isto permite chegar a um resultado bastante bom, quarto melhor algoritmo em termos de pontuação, mantendo um tempo de execução bastante baixo, 3 melhor algoritmo em termos de tempo de execução.

5. Tal como foi previsto anteriormente, ao correr o algoritmo **A*** com a propriedade do GoalState = 0, vão ser percorridos nós que de outra forma não seriam se GoalState = h*(n). Podemos analisar nos gráficos que isso tem a sua consequência, tanto em termos de tempo de pesquisa como de média de pontuação, que ambos aumentam o seu valor.

Conclusões Finais

Achamos que o trabalho foi realizado com um resultado positivo.

Com este trabalho conseguimos estudar e provar com auxilio de testes exaustivos que o algoritmo **A*** não é o mais indicado para resolver problemas de maximização de objetivos, ou pelo menos, não de uma forma eficiente.

Quando conseguimos provar isso sentimos necessidade de achar um novo algoritmo que fosse eficiente, de modo a dar ao nosso programa um algoritmo que fosse realmente motivador e estimulante ao ser comparado com um jogador humano.

Esparamos poder mostrar este trabalho nalgum momento extra curricular visto que estamos bastante orgulhosos do que conseguimos fazer.

Melhoramentos

No futuro podemos adicionar as seguintes funcionalidades:

1. Permitir analisar cada jogada do algoritmo passo a passo.
2. Criar e/ou carregar de memória um tabuleiro anteriormente concebido.
3. Registo de HighScores.

Recursos Utilizadas

Bibliografia

- Apontamentos das aulas teóricas.
- Artificial Intelligence: A Modern Approach : http://aima.cs.berkeley.edu

Webgrafia

- Wikipédia. Algoritmo A Star. Disponível em: https://pt.wikipedia.org/wiki/Algoritmo_A*
- Wikipédia. Algoritmo IDDA*. Disponível em: https://en.wikipedia.org/wiki/Iterative_deepening_A*
- TutorialsPoint. Algoritmo Depth First Search. Disponível em: http://www.tutorialspoint.com/data_structures_algorithms/depth_first_traversal.htm
- Max Is More than Min: Solving Maximization Problems with Heuristic Search. Disponível em : https://www.aaai.org/ocs/index.php/SOCS/SOCS14/paper/viewFile/8927/8890
- Swing Tutorial. Disponível em: http://zetcode.com/tutorials/javaswingtutorial/
- Dúvidas Gerais. http://www.stackoverflow.com/

Software

- Green Felt. Jogo Hopeless. Disponível em: http://greenfelt.net/hopeless
- Apache POI. Disponível em: https://poi.apache.org/