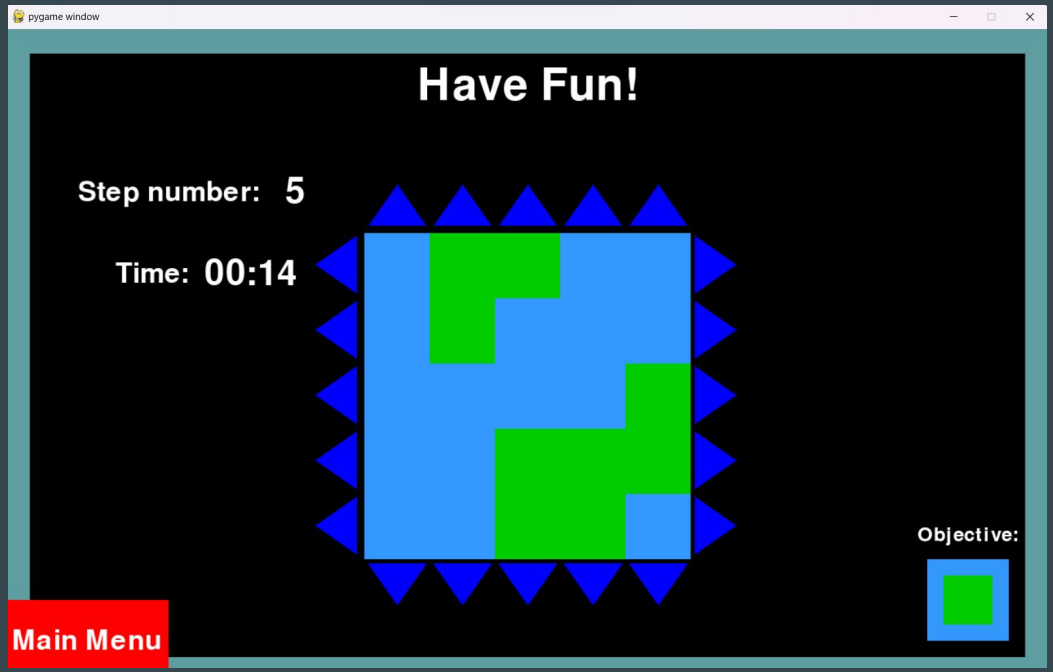
Inteligência Artificial

Presentation - Assignment 1

L.EIC029 - 2023/2024

Grupo A1_29
Francisco Lopes - up202108796
João Fernandes - up202108044
Rui Silveira - up202108878

{1} Cogito



> Cogito é um jogo de quebra-cabeça e estratégia que desafia a mente com desafios estimulantes.

> Com uma mecânica simples, mas envolvente, o objetivo é manipular elementos em um tabuleiro para alcançar um estado específico.

> Cada nível apresenta um novo desafio, de modo a testar as habilidades, o pensamento lógico e a estratégia do jogador.

{2} Formulação do Problema / {4} Heurísticas

> Estado Inicial (S0):

Representa o estado inicial do tabuleiro do jogo, no qual possui n^2 peças "golo", colocadas de forma aleatória.. Varia em: Organização de peças e Tamanho, consoante o nível de dificuldade.

> Representação do Estado(S):

Cada estado vai ter associado a si: um tabuleiro e o seu histórico de movimentos (isto é, quais (que também irá transformar-se em quantos) movimentos foram necessários para chegar a determinado tabuleiro).

> Operações (A):

As operações possíveis serão: movimentar uma coluna para cima ou para baixo e movimentar uma linha para a esquerda ou para a direita, em um número de casas (1), sem limitações, isto é, sem pré-condições (Ex: uma peça no fundo de uma coluna, caso se escolha movimentar para baixo essa coluna, essa peça irá para o topo da coluna). O custo é unitário, isto é, 1 passo.

> Heurísticas utilizadas:

H1 -> Esta heurística conta o número de peças que não estão na posição correta em relação ao estado objetivo. Quanto maior o número de peças fora de lugar, maior será o valor desta heurística, indicando que o estado atual está mais distante do estado objetivo.

H2 -> A distância de Manhattan é a soma das distâncias horizontais e verticais entre a posição atual de uma peça e sua posição final desejada no tabuleiro. Esta heurística calcula essa soma para todas as peças "golo" e retorna o resultado. Quanto menor a soma, mais próximo o estado atual está do estado objetivo.

> Branching Factor: N*4, sendo N o tamanho da matriz.

> State Space size, considerando n -> N^2 e m -> 9, sendo N o tamanho da matriz: $\frac{n!}{(n-m)!*m!}$

> Teste de Objetivo (GoalTest(S)):

As peças "golo" encontram-se todas em um quadrado no centro do tabuleiro..

{3} Algoritmos de Busca Implementados

Uninformed Search:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Depth-Limited Search (DFS com limite)
- Iterative Deepening Search (Busca iterativa com aprofundamento progressivo, ou seja, depth-limited search progressivo)

Informed Search:

- Greedy Search (com heurísticas)
- A* Search (com heurísticas)

{6} Algoritmos Usados

> Breadth-First Search (BFS):

A busca em largura é um algoritmo de busca que explora sistematicamente todos os nós vizinhos antes de passar para os nós vizinhos dos vizinhos. Começando pelo nó inicial, o BFS expande para todos os nós vizinhos do nível atual antes de seguir para os nós do próximo nível. Isso garante que a solução encontrada seja a mais próxima possível do nó inicial. O BFS é implementado usando uma fila para armazenar os nós a serem explorados.

> Iterative Deepening Search (IDS):

A busca com aprofundamento iterativo é uma estratégia que combina a busca em largura (BFS) com a busca em profundidade (DFS). Ela funciona realizando uma busca em profundidade limitada com uma profundidade máxima inicial e, se não encontrar a solução, aumenta progressivamente o limite de profundidade e tenta novamente. Isso é feito até que a solução seja encontrada. O IDS é útil em casos onde a profundidade máxima do espaço de busca não é conhecida antecipadamente.

> Greedy Search:

O algoritmo GREEDY faz escolhas baseadas apenas na informação disponível imediatamente, sem considerar o custo total até o objetivo. Ele seleciona o próximo nó com base em uma heurística que estima qual nó parece ser o mais promissor em direção ao objetivo. O GREEDY sempre escolhe o nó que parece mais próximo do objetivo, mesmo que isso possa levar a caminhos que eventualmente sejam ineficientes.

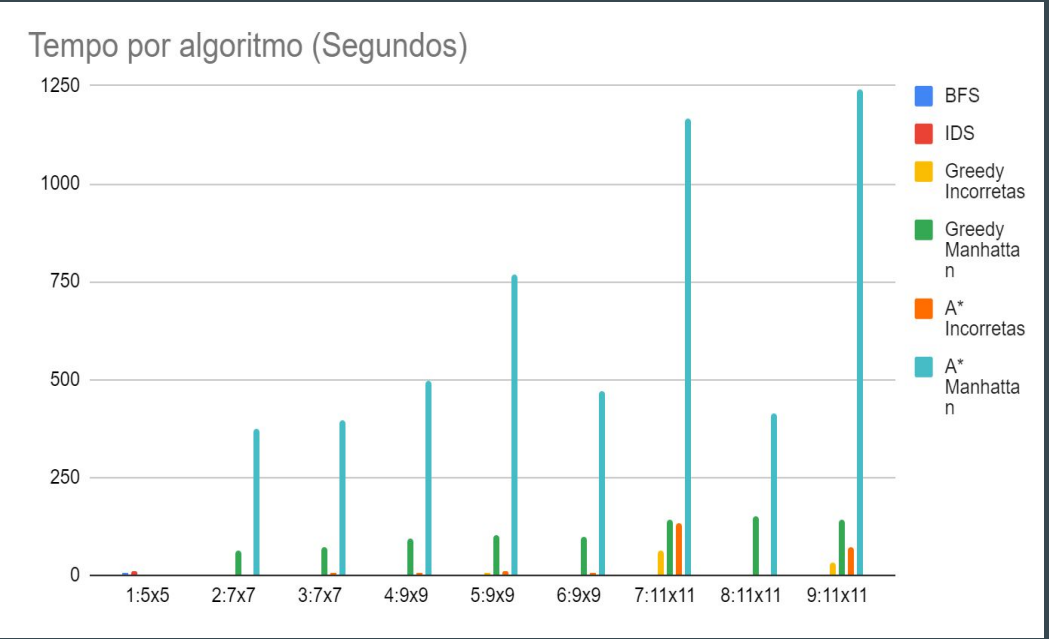
> A* Search:

O algoritmo A* é uma extensão do GREEDY que também leva em consideração o custo total do caminho até o nó atual. Ele combina o custo atual do caminho com uma estimativa heurística do custo restante até o objetivo para determinar a prioridade de cada nó. O A* seleciona o próximo nó com base na soma desses dois custos, escolhendo o nó que parece ser o mais promissor em termos de eficiência global.

Nota: DFS não foi utilizado no seu estado “primitivo” (isto é, sem depth limit) por simplesmente ser inviável, devido a ultrapassar o limite de recursão (branching factor e state space size elevados).

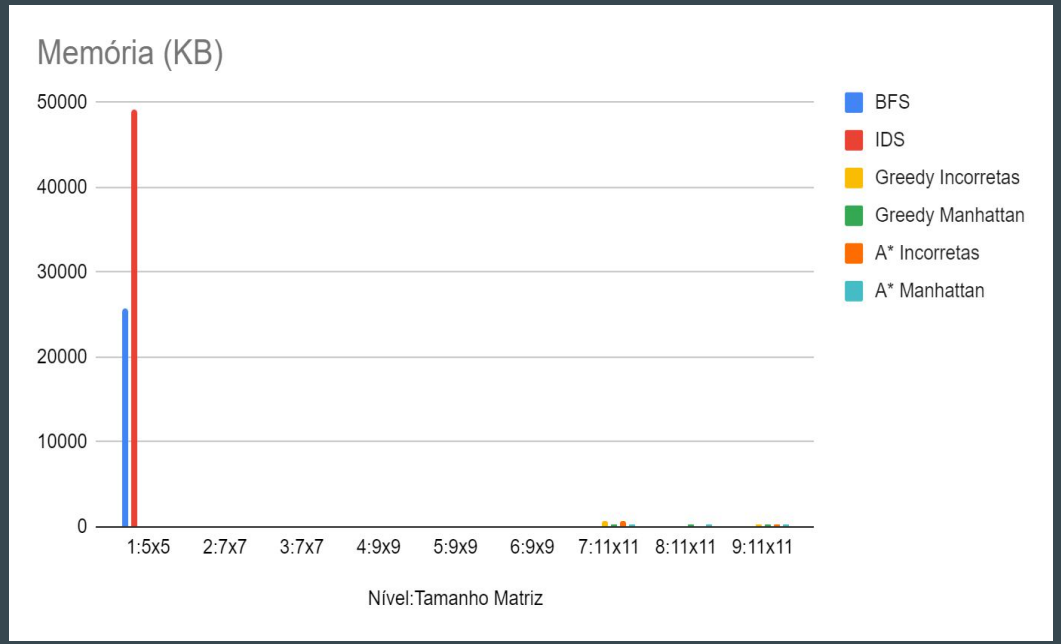
{8} Analysis

Este gráfico ilustra a eficiência temporal de diferentes algoritmos de pesquisa em diferentes tamanhos de matrizes do jogo "Cogito". É evidente que o BFS e o IDS são significativamente mais lentos, o que justifica a sua exclusão para matrizes maiores do que 5x5, pois tornam-se impraticáveis. O algoritmo Greedy com a heurística de contagem de peças incorretas mostra um desempenho inviável para a matriz de nível 8 11x11, indicando um aumento acentuado na dificuldade. Entretanto, o A* com distância de Manhattan mantém uma melhor escalabilidade no tempo, embora também mostre um aumento significativo nos puzzles maiores.



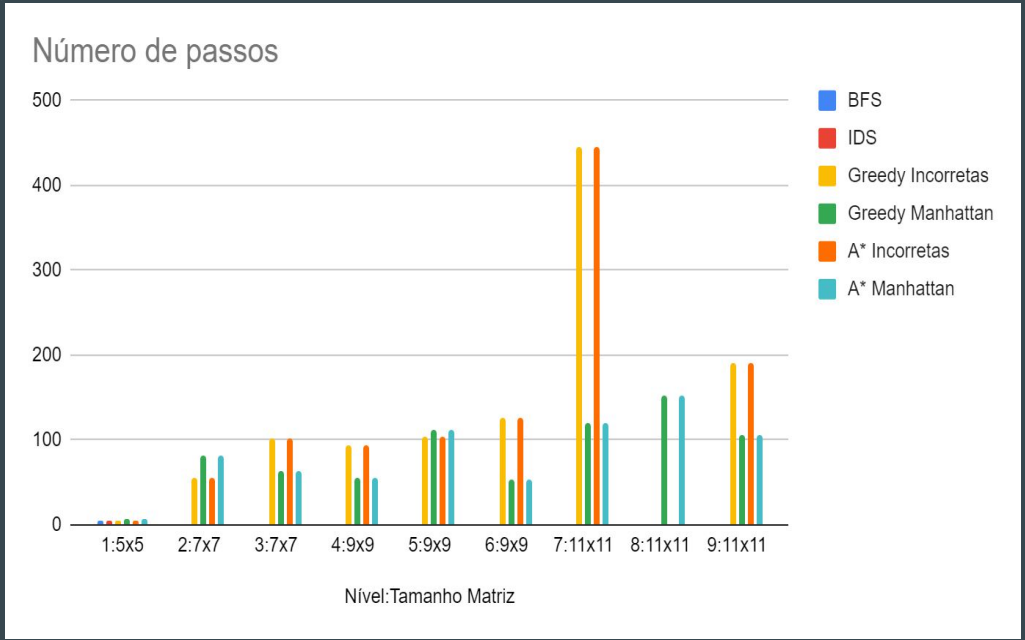
{8} Analysis

O gráfico do uso de memória sublinha a diferença acentuada no consumo de recursos entre os algoritmos de pesquisa. BFS e IDS mostram um uso de memória exponencialmente maior para o nível 5x5, sugerindo um fator limitante na sua aplicação prática a problemas maiores. Para os outros algoritmos, o uso de memória permanece relativamente baixo e consistente em todos os níveis, o que implica que, exceto para BFS e IDS, a memória não é uma restrição primária no desempenho destes algoritmos de pesquisa.



{8} Analysis

O BFS e o IDS, sendo pesquisas exaustivas, têm um número fixo de passos para a matriz 5x5, o que é consistente com a complexidade mínima deste nível. Para os algoritmos Greedy e A*, observa-se que o número de passos aumenta com o tamanho da matriz. Notavelmente, a pesquisa A* com a heurística da distância de Manhattan mantém um número de passos relativamente consistente em puzzles maiores, sugerindo que encontra soluções mais diretas. No entanto, o algoritmo Greedy com peças incorretas incorre num aumento significativo de passos para alguns puzzles maiores, potencialmente refletindo uma busca de caminho menos eficiente devido às limitações da heurística.



{9} Conclusion

O projeto aborda uma variedade de aspetos relacionados à implementação e análise de algoritmos de busca, com foco em um solitário de um jogador. Desde a definição dos estados inicial e objetivo até a formulação das operações possíveis e heurísticas de avaliação, cada aspeto foi cuidadosamente considerado para garantir uma abordagem abrangente e eficaz.

A implementação de algoritmos de busca *uninformed*, como a busca em profundidade iterativa e a busca em largura, ofereceu uma compreensão fundamental de como esses métodos exploram o espaço de busca de maneira sistemática. Por outro lado, a implementação de algoritmos de busca *informed*, como GREEDY e A*, destacou a importância de heurísticas eficazes na seleção de nós promissores e na otimização do processo de busca.

Além disso, o projeto demonstrou a aplicação prática desses conceitos por meio da resolução de um jogo específico, fornecendo uma plataforma para experimentar e comparar as diferentes estratégias de busca. Ao final, o projeto não apenas aprimorou o entendimento teórico dos algoritmos de busca, mas também oferece *insights* valiosos sobre a sua implementação e desempenho.

Podemos também a importância de ter em conta o branching factor e state space size na definição de problemas, pois valores elevados podem tornar a pesquisa ótima cada vez mais complicada e inviável.

Além disso, fomos capazes de observar a importância de heurísticas com capacidade de avaliar o tabuleiro, e a diferença que estas fazem.

{10} References

MYabandonware - https://www.myabandonware.com/game/cogito-1d4

old-games.com - https://www.old-games.com/download/3558/cogito

Youtube Gameplay - *Rubycored* - https://www.youtube.com/watch?v=xCP36Iaduss

Game Music - *kreepster05* - https://www.youtube.com/watch?v=w98eiaVfIro

Lecture Resources (Notebook, Practical Exercises...)

{10} Recursos usados

Linguagem de Programação: Python (PyGame)

Ambiente de Desenvolvimento: VSCode