1 UP 25000

2 UP 003200

COGITO

Inteligência Artificial
Projeto 1

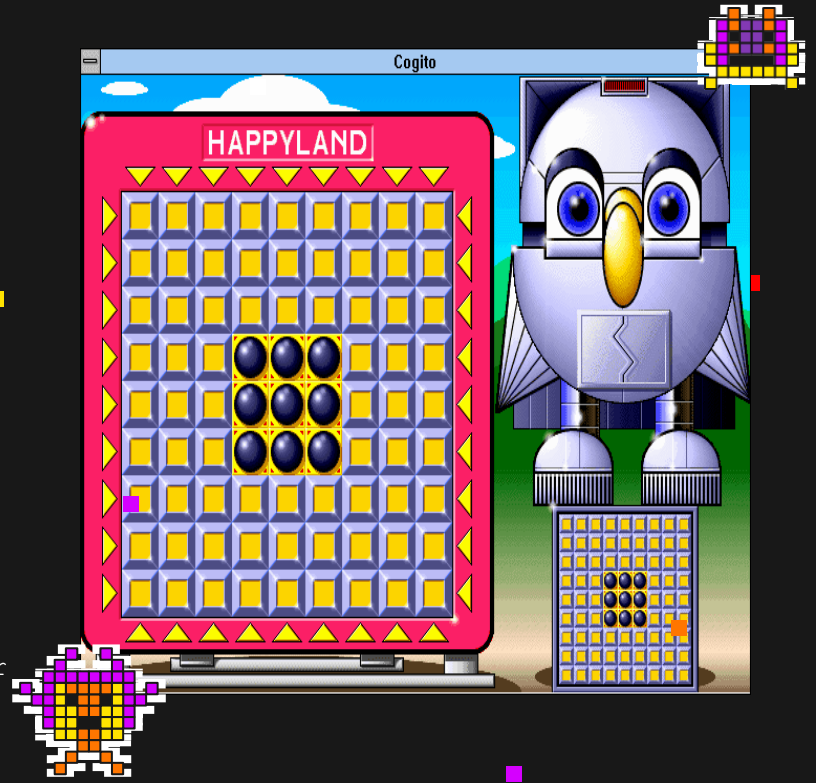
Ana Carolina Coutinho
José Costa
Afonso Poças

FEUP 23/24

CREDITS

Especificação do Problema

- O jogo "Cogito" é apresentado como um problema de otimização e busca.
- Neste quebra-cabeças lógico, o objetivo é reorganizar um conjunto de peças num tabuleiro para atingir uma configuração específica, utilizando o menor número de movimentos possíveis.
- Os jogadores são desafiados a pensar estrategicamente para encontrar a sequência de movimentos mais eficiente, tornando "Cogito" um excelente caso de estudo para algoritmos de busca e otimização em inteligência artificial.
- Este projeto tem como propósito desenvolver uma solução que, dado uma configuração inicial do tabuleiro, determine a sequência ótima de ações para resolver o quebra-cabeça, explorando conceitos como estados, operadores e heurísticas, a fim de minimizar o número de movimentos até a solução final.



CONTENTS OF THIS TEMPLATE

Nesta secção, apresentamos uma revisão de trabalhos relacionados que exploram o uso de técnicas de Inteligência Artificial (IA) em jogos de quebra-cabeças e otimização. Estes estudos fornecem insights valiosos para o desenvolvimento de soluções inovadoras para o jogo "Cogito", focando especialmente na otimização do nível de dificuldade e na aplicação de aprendizado por reforço.

Balancing Match-3 Puzzle Games with AI	Um estudo de Byounggwon Kim e Jungyoon Kim utilizou os algoritmos PPO e SAC para equilibrar a dificuldade em jogos Match-3. A análise revelou que o SAC é mais eficiente para ajustar a dificuldade dos estágios, indicando uma direção promissora para otimizar quebra-cabeças complexos, como "Cogito". https://www.mdpi.com/2079-9292/12/21/4456
Recent Research on AI in Games	Este artigo destaca a aplicação abrangente de IA no desenvolvimento de jogos, incluindo a geração de conteúdo e a otimização de níveis com base no comportamento do jogador. Enfatiza a importância da IA na criação de NPCs e no design de níveis, sugerindo melhorias na experiência do usuário em "Cogito". https://www.researchgate.net/publication/343244745_Recent_Research_on_AI_in_Games
Techniques and Paradigms in Modern Game AI Systems	Yunlong Lu e Wenxin Li revisam técnicas de IA em jogos, destacando o aprendizado profundo por reforço como promissor para jogos complexos, como "Cogito". Esta abordagem pode ajudar no desenvolvimento de agentes eficientes para resolver desafios do jogo. https://www.mdpi.com/1999-4893/15/8/282

Formulação do Problema

- **Representação do Estado**

Definição: Cada estado do jogo é representado pela configuração atual do tabuleiro, incluindo a posição de todas as peças.

Detalhe: O tabuleiro pode ser modelado como uma matriz bidimensional, onde cada célula representa um espaço que pode estar “vazio” ou ocupado por uma peça especial.

- **Estado Inicial**

Definição: O estado inicial do tabuleiro é a configuração no início do jogo, antes de qualquer movimento ser feito pelo jogador (e após as movimentações iniciais aleatórias do computador).

- **Teste de Objetivo**

Definição: Uma função que verifica se o tabuleiro atual corresponde à solução desejada (com todas as peças na região central).

Implementação: Comparação direta entre a configuração atual do tabuleiro e a configuração objetivo, considerando todas as peças nas posições corretas.

- **Operadores**

Movimentos Possíveis das Colunas:
Nomes: Mover para cima, mover para baixo, mover para a esquerda, mover para a direita.
Precondições: Apenas as colunas podem mover-se na vertical (cima ou baixo), e apenas as linhas podem mover-se na horizontal (esquerda ou direita).
Efeitos: As peças da coluna/linha "deslizam" no sentido escolhido, as peças no limite do tabuleiro passam para o lado oposto.
Custos: Cada movimento pode ter um custo uniforme (ex., 1 ponto por movimento).

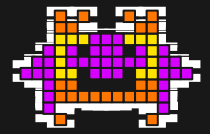
- **Heurísticas/Função de Avaliação:**

Distância de Manhattan: Calcular a soma das distâncias horizontais e verticais de todas as peças até as suas posições objetivas.

Número de peças fora do lugar: Conta quantas peças não estão na sua posição final desejada.

Esta formulação oferece uma base sólida para abordar o jogo "Cogito" como um problema de otimização e busca. A implementação destes conceitos permitirá a aplicação de algoritmos de busca, como busca A*, busca greedy para encontrar a solução ótima ou aproximada para resolver os quebra-cabeças propostos pelo jogo.

Trabalho de Implementação



01

Linguagem de Programação e Ambiente

Python e **Pygame**, integrados no ambiente de desenvolvimento Visual Studio: Optamos pela simplicidade e pela rica biblioteca gráfica do Pygame, utilizando a linguagem Python.

02

Estruturas de Dados

O tabuleiro é implementado por meio da classe **Board**, utilizando uma matriz bidimensional em Python **([[None for in range(size)] for in range(size)])**, para representar espaços vazios e peças (**'X'**).

03

Visualização e Interação

O objeto **Menu** é responsável pela interação do utilizador nas diferentes opções de menu.
A gameplay é gerida através das funções **handle_events**, **select_arrow** e **execute_move** da class **Game**

CREDITS

Algoritmos Usados

- **Procura Não Informada:**
  - BFS;
  - DFS;
  - Uniform Cost Search;
  - Iterative Deepening (Depth Limited Search).

- **Procura Informada:**
  - Greedy;
  - A*

- **Heurísticas Utilizadas:**
  - **Manhattan Distance:** Soma das distâncias respetivas de cada célula-alvo (‘X’) até a uma posição central (posições do Estado-Objetivo);
  - **Out-of-Place Cells:** Soma da quantidade de células-alvo fora das posições do Estado-Objetivo.

Resultados

		Algorithm							
		BFS	DFS	Uniform Cost	ative Deeper	A* Manhattar	eedy Manhat	* Out-Of-Pla	edy Out-Of-P
1 (3 initial m	me Average (	34,84	6,047	19,653	8,163	0,103	0,033	0,127	0,033
	Moves	3	3	3	3	3	3	3	3
2 (10 initial m	me Average (	-	-	-	-	0,92	0,633	7,617	0,357
	Moves	-	-	-	-	6	10	6	8
3 (20 initial m	me Average (	-	-	-	-	-	0,83	-	5,013
	Moves	-	-	-	-	-	18	-	19

- De modo a comparar os resultados obtidos, testamos os algoritmos em configurações iniciais do tabuleiro iguais, fazendo variar apenas o número de movimentos iniciais em cada (Nível 1 = 3 movimentos iniciais; Nível 2 = 10 movimentos iniciais; Nível 3 = 20 movimentos iniciais)

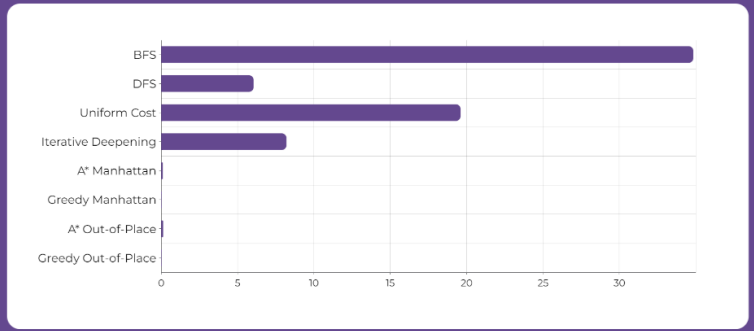
- Como podemos observar. Apenas o algoritmo greedy consegiu apresentar resultados numa janela de tempo razoável (<=60s) em todos os níveis.

Resultados

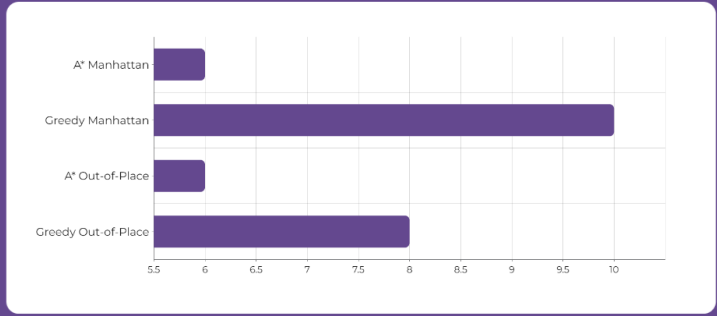
- No nível 1, o único resolvido por todos os algoritmos, a diferença de tempo entre os algoritmos de procura informada e não informada é muito grande. Já a quantidade de steps para encontrar a solução é igual para todos 3, o que possivelmente deve-se às reduzidas alterações ao tabuleiro inicial, que coincide com estado-objetivo.

- No nível 2, apenas os algoritmos de procura informada conseguem chegar à solução, destacando-se o Elevado tempo demorado pelo A* com a heurística out-of-place. Para além disto, é de salientar o reduzido número de steps usado pelo A* em comparação com o método greedy. O número ideal de steps estimado é o mesmo que o número inicial de movimentos aleatórios.

Tempo utilizado por cada Algoritmo (nível 1)



Número de Movimentos por Algoritmo (nível 2)



Conclusão

As procuras informadas revelaram-se superiores na resolução deste problema, nomeadamente a procura Greedy. Já as procuras não informadas revelaram-se inviáveis no contexto de jogos muito complicados. Isto deve-se provavelmente ao elevado branching factor do nosso problema, que é na maioria dos momentos aproximadamente 15. Deste modo, quando apresentados problemas que exigem uma depth de procura maior, como é o caso do nível 2 e 3, a procura não informada não resulta.

Referências Adicionais

- https://www.geeksforgeeks.org/searching-algorithms/ (see table of contents)
- https://www.pygame.org/docs/