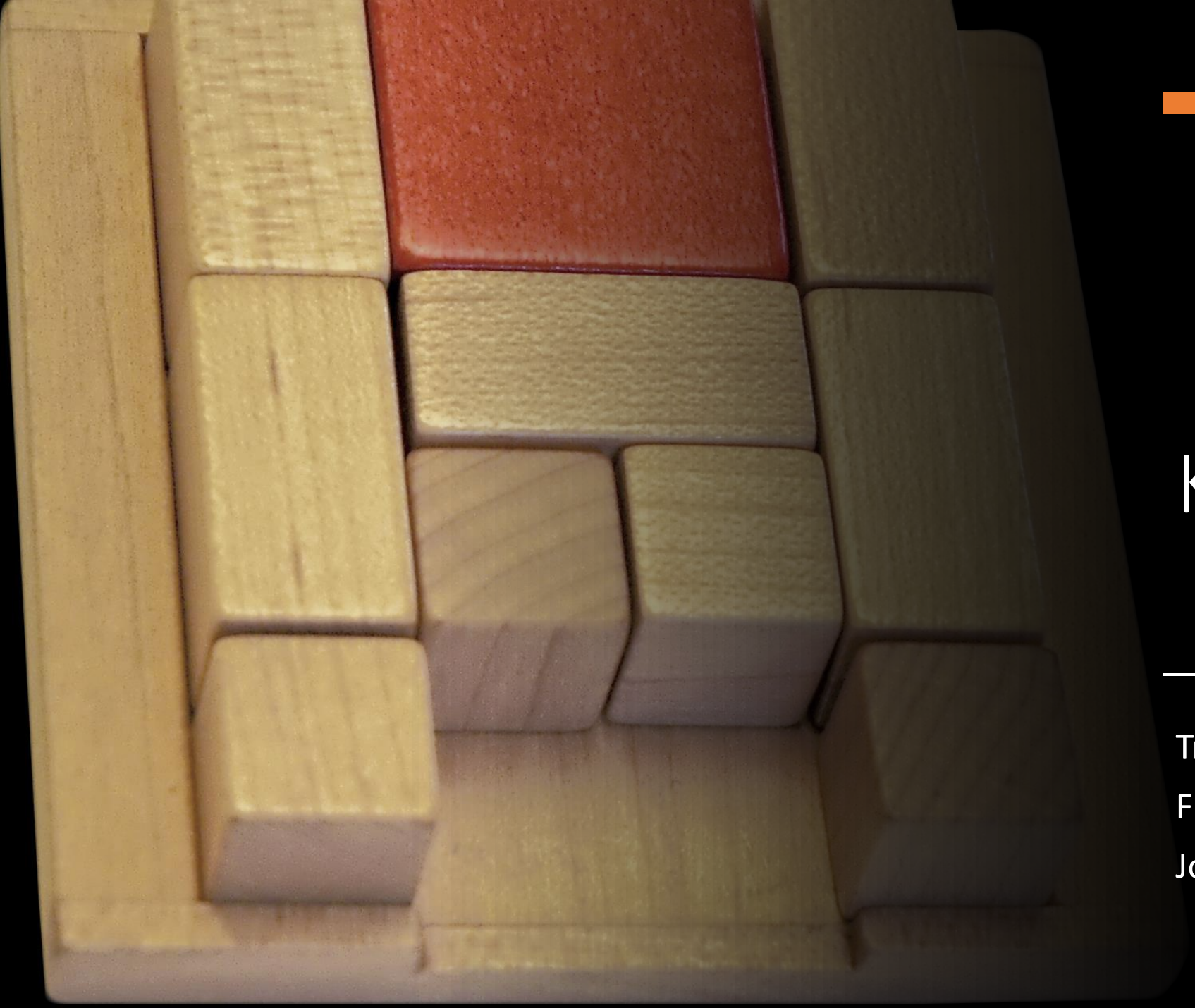


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Klotski

Trabalho de:

Francisco Martins 202107432

João Baptista 202207629

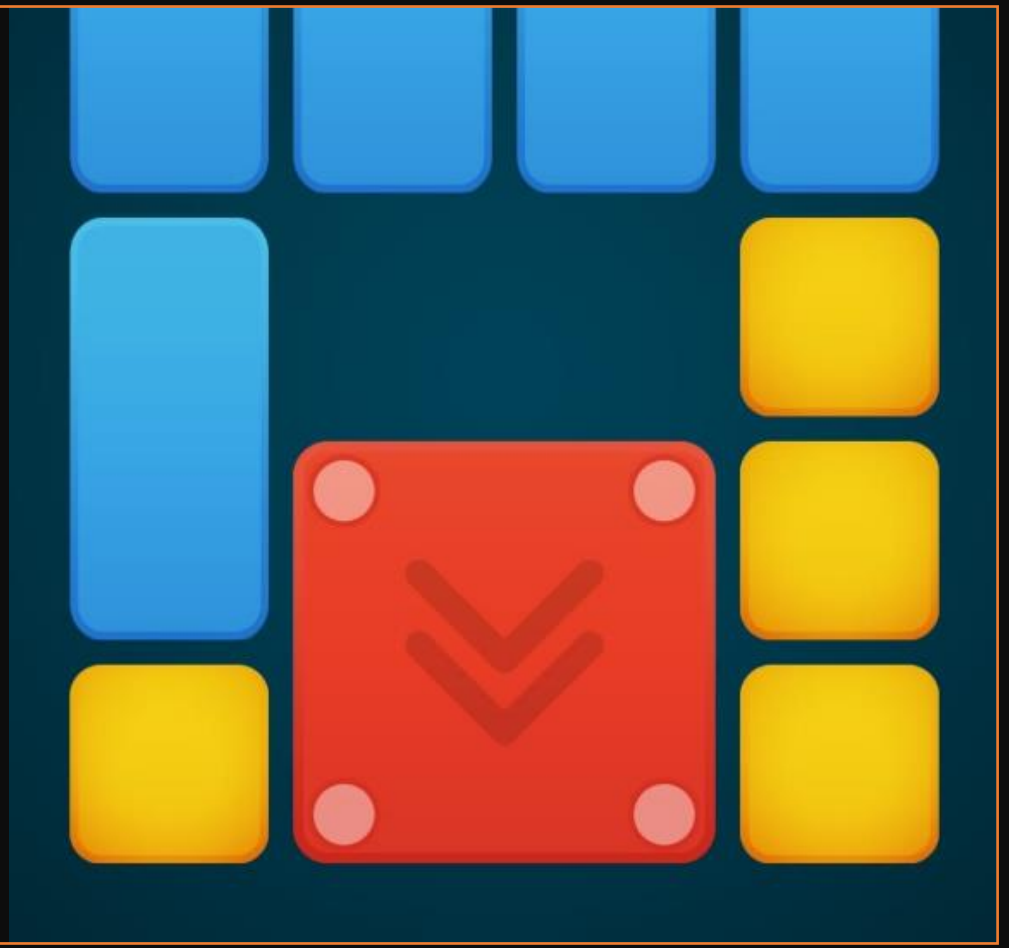
Elementos da Inteligência Artificial e Ciência de Dados 2022/2023

Processo de desenvolvimento

- Funcionamento do jogo
- Métodos de pesquisa
- Comparação entre métodos
- Desenvolvimento da interface

Funcionamento do Klotsky

- Objetivo de levar a peça para a posição centrada do tabuleiro na linha mais abaixo
- Não pode haver sobreposição de peças
- Mínimo de movimentos possíveis



Transformação num problema de pesquisa

- Definição dos estados
- Divisão do tipo de peças por valor
- Utilização de dicionário para ter minipeça representativa
- Estrutura do jogo em matriz
- Função de teste objetivo

```
#value of board;
#if value%2 == 0 and value != 0 ->double piece vertical
#if value%2 != 0 and value != 1 and value >0 -> double piece horizontaly
#if value < 0 -> sigle piece
#if value == 1 -> big piece *peça objetivo*
#if value == 0 -> free space
```

```
def test_goal(self):
    mid_of_board=len(self.board[0])//2-1
    if 1==self.board[-1][mid_of_board] and self.board[-1][mid_of_board+1]==1:
        return True
    return False
```

```
def dicionario(nivel_board):
    pieces = {}
    for i in range(len(nivel_board)):
        for j in range(len(nivel_board[i])):
            piece_id = nivel_board[i][j]
            if piece_id not in pieces and piece_id != 0:
                pieces[piece_id] = (i, j)
    return pieces
```

```
class Klostki:
    def __init__(self,board,pieces,move_history=[]):
        self.board = deepcopy(board)
        self.pieces=deepcopy(pieces)

        self.move_history = [] + move_history + [self.board]
```

```
|   | 3 | 3 |   |
-----------------
|   |   | 1 | 1 |
-----------------
|   |   | 1 | 1 |
-----------------
|   | 4 |   |   |
-----------------
|   | 4 | -2 |   |
-----------------
```

Métodos de Pesquisa

BFS:

- Garante o menor caminho possível
- Demora muito tempo

Greedy:

- Demora menos tempo que a BFS
- Não garante o melhor caminho

A*:

- Garante o menor caminho
- Grande probabilidade de ser mais rápido que a Greedy e que a BFS

```python
def bfs(problems):
    queue = [problems]

    while queue:
        board = queue.pop(0)

        if board.test_goal():
            resultados=board
            break

        for child in board.children():

            queue.append(child)


    return resultados
```

```python
def greedy_search(problem, heuristic):
    setattr(Klostki, "__lt__", lambda self, other: heuristic(self) < heuristic(other))
    states = [problem]
    visited = set()
    while states:
        current=heapq.heappop(states)
        visited.add(current)
        if current.test_goal():
            return current

        for child in current.children():
            if child not in visited:
                heapq.heappush(states, child)
    return None
```

```python
def a_star_search(problem, heuristic):
    return greedy_search(problem, lambda state: heuristic(state) + len(state.move_history) - 1)

print_sequence(a_star_search(problems(1), h1))
```

Heurísticas e Comparação de Métodos de pesquisa

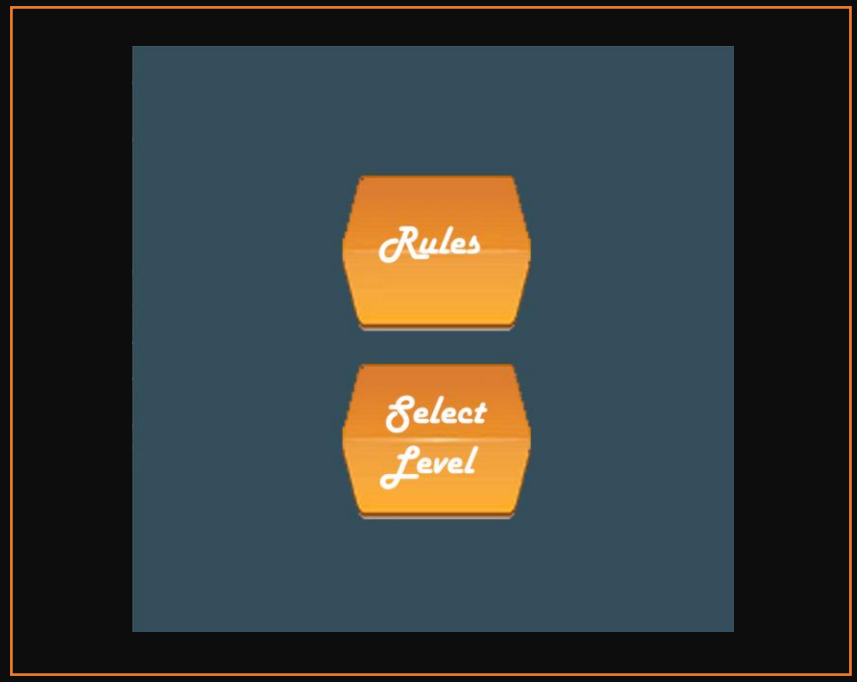
```python
def h1(board):
    mid_of_board=(len(board.board[0])//2)-1
    if board.pieces[1]==(len(board.board)-2,mid_of_board):
        return 0
    else:
        peças = []
        for i in range(-2,0):
            for j in range(mid_of_board,mid_of_board+2):
                if board.board[i][j] != 0 and board.board[i][j] != 1 and board.board[i][j] not in peças:
                    peças.append(board.board[i][j])
        return len(peças)+1

def h2(board):
    mid_of_board=(len(board.board[0])//2)-1
    row,col = board.pieces[1]
    h2 = abs(len(board.board)-2-row) + abs(mid_of_board-col) +(h1(board)-1)
    return h2
```

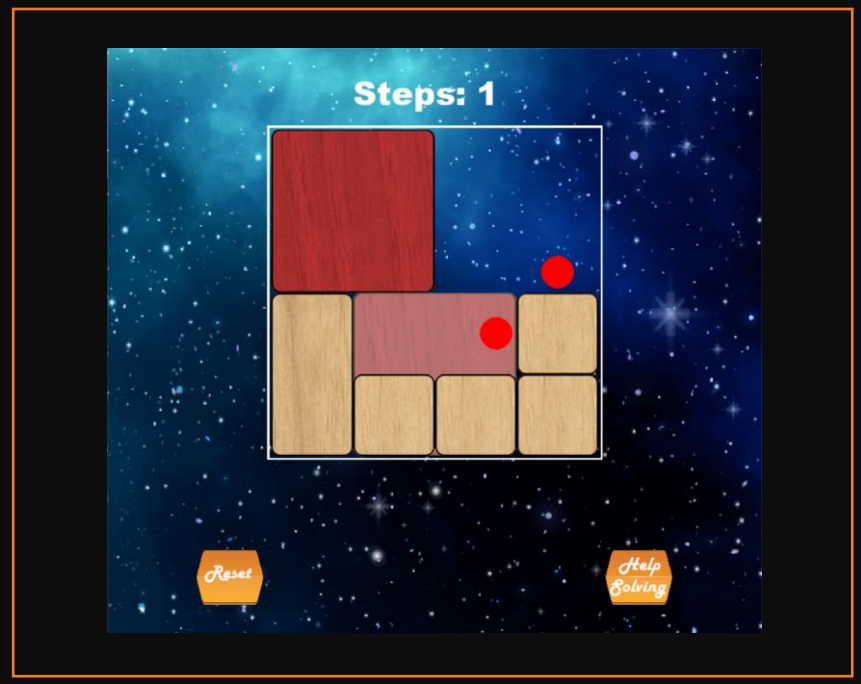
- A h1 retorna 0 caso a peça principal esteja no seu destino (objetivo concluído, caso contrario retorna um contador que é inicializado com o valor porque a peça principal já está fora do sitio e cujo valor é incrementado em uma unidade por cada peça que esteja a ocupar o lugar onde deveria estar a peça final
- H2 retorna o número de paços que levaria á peça principal para chegar ao objetivo, caso ela fosse a única peça no tabuleiro
- Quer na Greedy search quer na A* a h2 provou-se muito mais eficaz resolvendo o problema com menos paços e em menos tempo

	VELOCIDADE / PASSOS			
	Teste 1	Teste 2	Teste 3	Teste 4
BFS	3min 15,4s/7steps	30+min	30+min	30+min
Greedy(h1)	0,0s/7steps	1.8s/60steps	17,4s/66steps	0,1s/15steps
Greedy(h2)	0,0s/7steps	0,2s/30steps	0,0s/18steps	0,1s/19steps
A*(h1)	0,7s/7steps	30+min	30+min	36,1s/13steps
A*(h2)	0,1s/7steps	8min 6,6 s/ 12steps	2min20,9s/12steps	17,0s/13steps
	Teste 5	Teste 6	Teste 7	Teste 8
BFS	30+min	30+min	30+min	30+min
Greedy(h1)	30+min	30+min	30+min	7,6s/23steps
Greedy(h2)	30+min	1min52s/21steps	1,3s/16steps	0,5s/13steps
A*(h1)	15,8s/44steps	30+min	30+min	3min53s/8steps
A*(h2)	1,7s/13steps	7min3s/11steps	30+min	1,7s/8steps

Desenvolvimento da Interface

1. Movimentação das peças
2. Esclarecimento das Regras
3. Opção de escolha do nível de maior ou menor
4. Opção de pedir ajuda a IA para resolver





```python
class Klostki:
    def __init__(self,board,pieces,move_history=[],escolhida=None,possible_moves=[]):
        self.board = deepcopy(board)
        self.pieces=deepcopy(pieces)
        self.move_history = [] + move_history + [self.board]
        self.escolhida= escolhida
        self.possible_moves= possible_moves
```

```python
def children(self):
    functions = [Klostki.move_up, Klostki.move_down, Klostki.move_left, Klostki.move_right]
    children = []
    for func in functions:

        for piece in self.pieces:
            temp=deepcopy(self)
            child = func(temp,piece)
            if child:
                child.move_history.insert(-1,self.board)
                children.append(child)
    return children
```

```python
def mover(self,p,row,col):
    if self.pieces[p][0]-1==row and self.pieces[p][1]==col:
        return self.move_up(p)
    if self.pieces[p][0]==row and self.pieces[p][1]-1==col:
        return self.move_left(p)
    if p>0 and (p==1 or p%2==0):
        if self.pieces[p][0]+2==row and self.pieces[p][1]==col:
            return self.move_down(p)
    elif self.pieces[p][0]+1==row and self.pieces[p][1]==col:
        return self.move_down(p)
    if p >0 and (p==1 or p%2!=0):
        if self.pieces[p][0]==row and self.pieces[p][1]+2==col:
            return self.move_right(p)
    elif self.pieces[p][0]==row and self.pieces[p][1]+1==col:
        return self.move_right(p)
```

Mudanças no código

1. Adicionamos parâmetros há Class para saber se o jogar tinha selecionado alguma peça e para saber quais os moves possíveis para essa mesma peça

2. Alteramos a children() com um deepcopy para distinguir as childs da root, pois ao usar função de mover ela irá alterar o objeto usado

3. Acrescentamos a função mover() para tornar iterativo ao utilizador mover as peças com o uso do rato

Ajuda da AI na resolução

Caso o utilizador tenha dificuldade em encontrar a solução do problema poderá clicar no botão “Help solving” que fará com que o programa recorra a um método pesquisa para descobrir um solução

- Escolha do método A*
    1. Encontra sempre a melhor opção
    2. É consideravelmente mais rápido que a BFS

Fontes e Webgrafia

- Ficha Prática 2
- EIACD_Lecture3_SolvingSearch.pdf
- Chat GPT
- https://www.youtube.com/watch?v=jO6qQDNa2UY&t=4682s