

# SOKOBAN AGENT

Unidade Curricular de Inteligência Artificial 2020/2021  
Professores Luís Seabra Lopes e Diogo Gomes

Alexandra Carvalho nmec 93346  
Anthony Pereira nmec 93016  
Luís Valentim nmec 93989

# Arquitectura

```
class tree_search.MyTree
  __init__(self, problem)
  get_plan(self, node)
  isRepeatedState(self, node)
  search(self)
  problem
  solution
  open_nodes
  visited_states
```

```
class tree_search.MyNode
  __init__(self, state_map, parent, depth, heuristic, action)
  __str__(self)
  __repr__(self)
  __lt__(self, other)
  parent
  depth
  heuristic
  action
  state_map
```

```
class tree_search.MyProblem
  __init__(self, domain)
  goal_test(self, state)
  initial
  domain
```

```
class tree_search.MyMap
  __init__(self, mapa)
  filter_tiles(self, list_to_filter)
  keeper(self)
  boxes(self)
  empty_goals(self)
  set_tile(self, pos, tile)
  clear_tile(self, pos)
  ver_tiles
  _keeper
  mapa
  hor_tiles
```

```
class tree_search.MyDomain
  __init__(self, initial)
  cornerCheck(self, state_map, pos)
  BoxesNextToWall(self, state_map, pos_init, pos)
  BoxNextWallNotGoal(self, state_map, pos_init, pos)
  deadlocks(self, state_map, pos_init, pos)
  actions(self, state_map)
  result(self, state_map, action)
  satisfies(self, state_map)
  heuristic(self, state_map)
  initial
```

# Arquitetura

1

## Domínio: Prevenção de Deadlocks e Detecção de Ações Possíveis

Search tree (MyTree -> linhas 423-464), baseada nas desenvolvidas durante as aulas práticas, contém nodes (MyNode -> linhas 401-416) representativos de jogadas possíveis. Para diminuir o número de nodes que é necessário analisar, são utilizadas funções de deadlock que eliminam nodes que impossibilitem encontrar a solução (MyDomain -> linhas 75-278).



Deadlock 1: cantos  
(caixa com paredes em 2 lados consecutivos)



Deadlock 2: caixas consecutivas na parede  
(caixa com caixa e parede em 2 lados consecutivos)



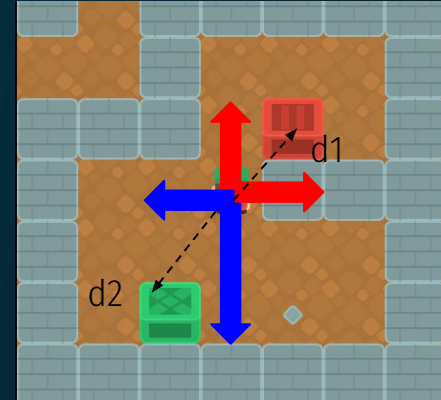
Deadlock 3: eixo bloqueado pela parede  
(caixa bloqueada pela parede num eixo sem goal)



# Arquitetura

2

## Árvore: Ciclo Principal de Pesquisa e Heurística



Na classe `MyTree` estão definidos uma lista de nós por explorar (inicializada com o nó `root`), uma lista de estados visitados e, por fim, uma solução; esta classe contém as seguintes funções:

- linhas 431-436: `get_plan(node)` → função chamada pela função `search()` quando uma solução é encontrada, e que devolve uma lista das ações efetuadas pelo keeper desde o nó-raiz até ao nó-solução;
- linhas 438-441: `isRepeatedState(node)` → para melhoria de performance do algoritmo, verifica se estado (mapa) do nó;
- linhas 445-464: `search()` → função onde se realiza a pesquisa de uma solução. Como ordena a lista dos nós “abertos” pela heurística, adota o algoritmo “greedy”. A heurística utilizada foi baseada na distância de Manhattan (entre o keeper e a caixa mais próxima) e o número de Goals por preencher.

# Resultados e Conclusões

intel Core i5 10th Generation / 8GB RAM -> Bloqueia no nível 106 -> 4980958 pontos

intel Core i7 8th Generation / 16GB RAM -> Bloqueia no nível 106 -> 4978828 pontos

intel Core i7 8th Generation / 8GB RAM -> Bloqueia no nível 68 -> 3262702 pontos

## Aspetos positivos:

- Utilização de um dicionário que guarda os estados já visitados, evitando explorar um caminho já explorado
- Utilização da heurística já analisada
- Implementação dos 3 deadlocks mais comuns
- Melhoria da performance recorrendo a dicionários, sets, list comprehensions, declaração de variáveis locais e “avoiding dots”

## Aspetos a melhorar:

- Ter duas tree searches, uma para o keeper e outra para as caixas
- Melhorar a heurística
- Implementar freeze deadlocks
- Representação do mapa num formato mais leve, por exemplo mapear os tiles com bits
- Melhorar a performance