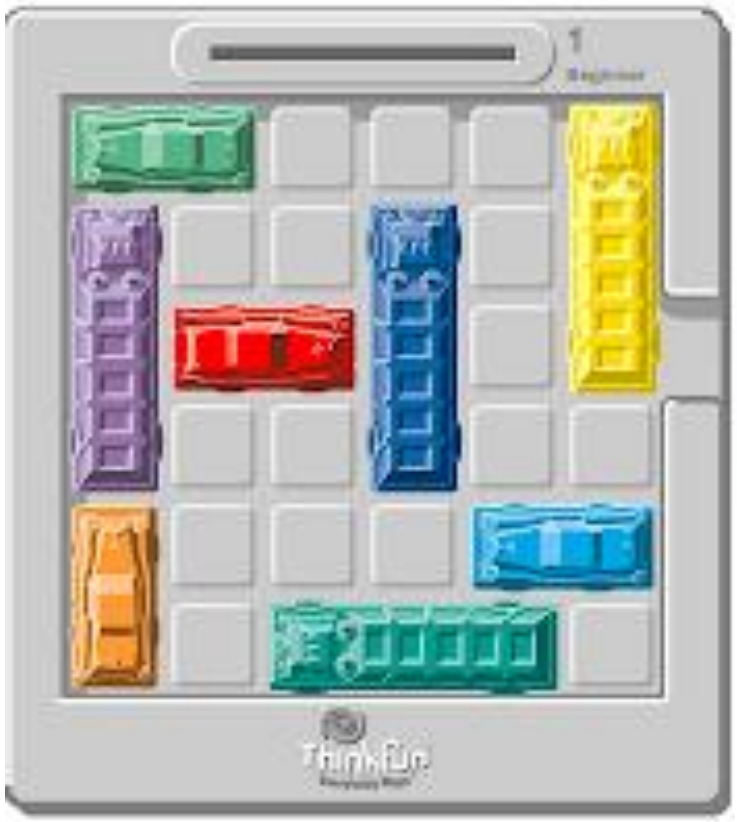
RUSH HOUR



DETI –UNIVERSIDADE DE AVEIRO

IA- INTELIGÊNCIA ARTIFICIAL

PROF. DIOGO GOMES,

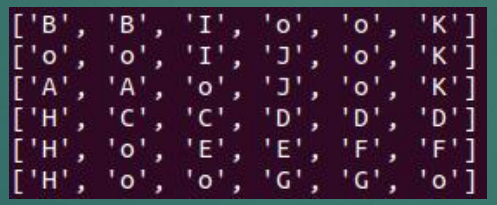
PROF. LUÍS SEABRA LOPES

PROF. AYMAN RADWAN

Sebastian D. González (103690)
Bárbara Moreira (104056)

Breve Introdução:

- O nosso projeto baseia-se na implementação de um Rush Hour Game.
- Trata-se de um jogo de lógica e estratégia com o intuito de encontrar uma saída para um carro vermelho movendo os carros que estejam em seu redor.
- Para a construção do mesmo é nos dado diversos níveis e, dependendo dos mesmos, o tabuleiro (implementado como uma grelha) poderá apresentar diferentes dimensões (4x4, 6x6 ou 8x8).
- Esta dimensão é calculada a partir da função grid localizada no início do student.py e que também nos permite saber o formato do nosso nível, ou seja, quais são as peças e os carros que o constituem o tabuleiro armazenado os no seguinte formato:



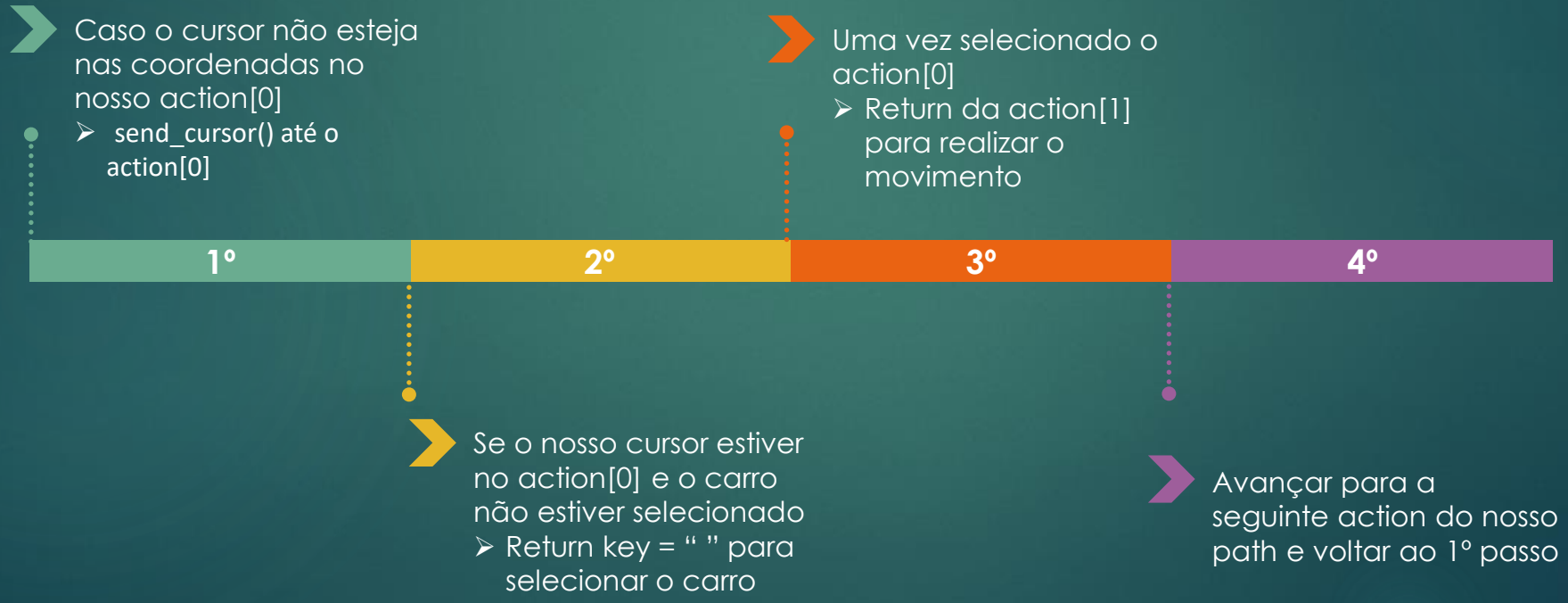
- Conhecendo o tabuleiro aplicamos o nosso algoritmo de pesquisa binária que nos devolve um tuplo constituído pelas actions necessárias para passar o nível atual:

[('A', 'd'), ('A', 'd'), ('A', 'd'), ('A', 'd')]

- action[0] representa a peça que queremos mover e action[1] a key a executar o movimento calculado.

Classe Agent

- Para resolver os nossos níveis é necessário percorrer todas as *actions* do nosso *path* e executá-las uma a uma até o nosso carro "A" chegar ao seu destino *(goal)*.
- Este processo é executado na função run_solver(), o methodo principal desta classe, onde é chamado o nosso algoritmo de pesquisa.
- O seguinte diagrama mostra de forma resumida o funcionamento da nossa função run_solver().



A pesquisa:

- Para o bom funcionamento do nosso jogo, tornou-se imprescindível a implementação de um algoritmo capaz não só de passar o máximo de níveis possível mas também fazê-lo da maneira mais inteligente e eficiente.
- O nosso algoritmo encontra-se implementado no ficheiro search.py, dividido em duas classes: ***SearchNode*** e ***SearchTree***.
- A classe principal é a ***SearchTree***. Esta apresenta diversos métodos para a ajudar na pesquisa. O método *heuristic()* calcula a distância entre o carro vermelho e a saída e itera sobre as colunas da *grid* à direita do mesmo, verificando se há carros que bloqueiem o caminho. Esta função foi essencial para melhorar o nosso algoritmo pois a pesquisa em si é feita no método *search()*, que recorre à heurística para assegurar que os nós com menos custo e maior valor são explorados primeiro.
- A classe ***SearchNode*** representa os estados (nós) na classe ***SearchTree***.

Outras Funções

solver.py

get_coordinates_cars()

Encontra as coordenadas de um dado carro na grid. Itera sobre a grelha e verifica se o elemento correspondente é o carro, se for retorna as suas coordenadas.

send_cursor()

Determina a direção para a qual o cursor se deve mover.

orientation()

Determina se o carro é vertical ou horizontal, isto afeta a forma como é movimentado na grelha.

search.py

get_path()

Método recursivo usado para obter a sequência de movimentos que levam ao estado final. Começa por verificar se o estado atual é o ultimo, se for, cria uma lista vazia de moves e retorna essa lista, caso contrário a função é novamente chamada.

change_grid()

Usado para gerar novos estados. Recebe o nó atual, a direção para mover um dado carro e os nós novos já gerados (lnewnodes).