



universidade
de aveiro

IA-Rush Hour

Realizado por:

- Pedro Rasinhas NMEC : 103541
- Gonalo Silva NMEC : 103668

AGENTE


> Iniciamos duas listas, *actions_list* e *key_list*, onde vamos armazenar as ações e as keys (para cada ação) respetivamente.

> A solução para um determinado nível, ou seja, o conjunto de ações necessárias para passar esse nível é armazenado na lista *actions_list*. A partir desta lista, retiramos a primeira ação e, para essa ação, calculamos as *keys* necessárias para a satisfazer, sendo estas armazenadas na lista *key_list*.

> Tendo as *keys* para uma ação, damos *pop* da primeira e enviamos para o servidor, até a *key_list* ficar vazia. Nesse momento calculamos as *keys* para a ação seguinte até não termos mais ações (nível completo).

```
state = json.loads(  
    await websocket.recv()  
) # receive game update, this must be called timely or your game will get out of sync with the server  
  
# verify if a crazy car happened  
if (ib != state.get("grid").split(" ")[1] and flag_nl) or not flag_happened: ...  
  
# if a crazy car happened  
if crazy_car: ...  
  
# if ainda não resolvemos a solução and se ainda nao temos as keys para uma action  
if action_list == [] and key_list == []: ...  
  
# ja temos as actions, mas ainda nao mandamos as keys para a action  
elif len(action_list) > 0 and (len(key_list) == 0): ...  
  
# ja temos as keys para a action, mas ainda nao mandamos todas as keys  
if len(key_list) > 0: ...
```

Crazy Car & Server *Desync*



> A nossa implementação para resolver os crazy cars baseia-se em recalcular a solução para o novo estado (após o crazy car).

> Como só calculamos o estado suposto após uma ação, excluindo, nesse cálculo, a possibilidade de ocorrência de um crazy car após enviarmos todas as keys (para conseguirmos saber se o movimento de facto ocorreu/foi possível) pode haver ocorrências de crazy cars, enquanto a lista de keys não está vazia, o que faz com que o fluxo de funcionamento do agente “rebente” por alguns momentos.

> Posto isto, como o cálculo de algumas soluções é demorado ($> 0.1s$), pode causar com que fiquemos dessincronizados com o servidor. Para prevenir este *desync* nós calculamos o tempo que demora a calcular a lista de ações (solução) para o nível em questão, e baseado nesse tempo (*tts*) fazemos:

```
if tts > 0.1:
    nit = round(tts*10)
    [await websocket.recv() for _ in range(nit)]
```

Deste modo, estamos sincronizados com o servidor e podemos continuar a execução.

HEURÍSTICA

Heurística usada → distância entre o nosso veículo até à saída + o número de carros a bloquear o caminho até à saída.

1→criar variável “board”(lista de listas)

2→car_block_num = num de carros a bloquear a saída

3→percorrer o board até encontrar o carro com Id “A” podendo assim calcular a distância até à saída (distância).

Ao encontrar “A” ficámos a saber a linha da tabela onde temos de contar os carros a bloquear e fazemos isso percorrendo as diferentes colunas nessa linha e ver quais campos diferem de “o”(campo livre).

Cálculo

```
def heuristic(self, state, goal):  
    # Distance from OUR vehicle to exit + number of cars blocking the way  
    board = [  
        list(state[i: i + self.width]) for i in range(0, len(state), self.width)  
    ]  
    car_block_num = 0  
  
    for r in range(self.width):  
        for c in range(self.width):  
            #r is rows, c is columns  
            if board[r][c] == "A":  
                distance = self.width - 1 - c  
                # check for cars blocking the way  
                for i in range(c + 1, self.width):  
                    if board[r][i] != "o":  
                        # print("car block: ", board[r][i])  
                        car_block_num += 1  
                return distance + car_block_num
```



Tree Search & Bibliografia

- > A pesquisa em árvore é baseada na que foi trabalhada ao longo do semestre nas aulas práticas.
- > Adaptámos o domínio, e fizemos alguns *improvements* nas funções desenvolvidas ao longo das aulas.
- > Da pesquisa, retornamos a lista de ações para cada nível (em dicionários), que posteriormente serão usadas no agente.
- > Decidimos usar a pesquisa *greedy* que foi a que demonstrou melhor desempenho em comparação a *Breadth* e mesmo à pesquisa A*.
- > Nos nossos testes o A* demorava entre 5 a 10s mais tempo a encontrar a solução de todos os níveis.

Em 10 execuções obtivemos, em média:

> 1 555 224 pontos.

Bibliografia

<https://abbashommadi.github.io/AI-for-Rush-Hour-Game/>

Menções

Deixar um agradecimento ao colega André Butuc, que nos ajudou a compreender o porquê de estarmos a entrar em *desync* com o servidor. Obrigado André! You rock!