Projecto Inteligência Artificial (LEIC 3^o Ano, 1^o Semestre 2019/2020)

Dúvidas : manuel.lopes@tecnico.ulisboa.pt

1 Jogo : Scotland Yard

Neste jogo um grupo de 3 detectives (SrGuarda) tenta apanhar um ladrão (ZeTelhado) nas ruas de Londres. Existem 4 meios de transporte, taxi, autocarro, metro e barco. A cada jogada os detectives podem deslocar-se usando apenas um meio de transporte (não é possível ficar parado no mesmo lugar, e não podem usar o barco), após o movimento dos detectives o ladrão pode mexer-se usando qualquer meio de tranporte. A todo o momento todos os jogadores vêem as posições dos outros. Cada vez que mexem os detectives têm de usar bilhetes de transporte. Existem bilhetes de 4 metro, 8 de autocarro e 12 de taxi. É sempre obrigatório mexer e não é possível dois detectives ocuparem a mesma casa.

Figure 1: Mapa Scotland Yard (pocket edition)

O jogo termina. Vitória dos detectives se conseguirem apanhar o ladrão deslocando-se para a mesma casa em que ele se encontra. Vitória do ladrão se conseguir fugir até que os detectives já não tenham mais bilhetes para se deslocarem.

1.1 Bibliografia e ambiente de desenvolvimento

A matéria teórica necessária ao desenvolvimento do projecto pode ser encontrada no livro de texto adoptado [?]. O projecto deverá ser implementado em Python.

2 1a Fase

Nesta primeira fase vamos estudar métodos de procura para encontrar caminhos rápidos entre localizações. Será necessário implementar o código numa classe usando o seguinte interface:

```
class SearchProblem:
  def __init__(self, goal, model, auxheur = []):
pass
  def search(self, init, limitexp = 2000, limitdepth = 20, tickets = [math.inf,math.inf,math.inf]):
pass
```

- goal - lista com as localizações objectivo
- model - grafo com as transições entre localizações (fornecido)
- auxheur - lista com as coordenadas de cada localização (fornecido)
- init - lista com as localizações iniciais
- limitexp - limite de expansões
- limitdepth - limite de profundidade
- tickets - limite de bilhetes

A função *search* deverá retornar um lista com comprimento igual ao caminho realizado pelos agentes durante a procura. Cada elemento deverá conter duas listas uma com a lista de acções (transporte usado) e um outra lista com as localizações destino.

Exemplo considerando agentes inicialmente nas posições $[30, 40, 109]$ e com o objectivo de chegar a $[63, 61, 70]$.

```
SP = SearchProblem(goal = [63,61,70], model = U, auxheur=coords)
I = [30,40,109]
nn = SP.search(I,limitexp = 3000, limitdepth = 10, tickets = [5,20,2])
```

O resultado da procura é a lista [[[], [30, 40, 109]], [[2, 0, 1], [69, 41, 84]], [[0, 0, 1], [63, 42, 61]], [[0, 1, 1], [69, 60, 84]], [[1, 1, 0], [63, 61, 70]]]

Em que o segundo elemento [[2, 0, 1], [69, 41, 84]] indica que:

- o primeiro guarda usou o metro para ir para a posição 69
- o segundo guarda usou o taxi para ir para a posição 41
- o terceiro guarda usou o autocarro para ir para a posição 84

2.1 (2 val) Exercício 1 - Um agente (sem limite de bilhetes)

Neste exercício só queremos encontrar a solução para um agente, ignorando o limite de bilhetes.

2.2 (4 val) Exercício 2 - Um agente (com limite de bilhetes)

Neste exercício só queremos encontrar a solução para um agente, mas tendo em conta o limite bilhetes para cada tipo de transporte.

2.3 (6 val) Exercício 3 - Três agentes (sem limite de bilhetes)

Neste exercício queremos encontrar a solução para três agentes, ignorando o limite bilhetes. Relembrar que os diferentes agentes não podem estar simultaneamente na mesma posição. O estado final deverá ser atingido respeitando a ordem dada.

2.4 (4 val) Exercício 4 - Três agentes (com limite de bilhetes)

Neste exercício queremos encontrar a solução para três agentes, mas tendo em conta o limite bilhetes para cada tipo de transporte. Relembrar que os diferentes agentes não podem estar simultaneamente na mesma posição. O estado final deverá ser atingido respeitando a ordem dada.

2.5 (4 val) Exercício 5 - Três agentes (com limite de bilhetes)

Neste exercício queremos encontrar a solução para três agentes, mas tendo em conta o limite bilhetes para cada tipo de transporte. Relembrar que os diferentes agentes não podem estar simultaneamente na mesma posição. Neste caso, e visto que para apanhar o ladrão não importa a identidade do guarda, o estado final é atingido se todas as posições finais estiverem ocupadas independentemente da ordem.

2.6 (2val) Eficiência

Algums dos exercícios poderão demorar muito tempo a correr para encontrar a solução optima. É no entanto necessário implementar métodos que melhorem a eficiência. Todos os testes têm de correr em menos de 60 segs. Haverá um bónus de 2 valores para os 10 grupos mais rápidos e 1 valor para os 10 grupos seguintes.

3 Entregas e Prazos

A realização do projecto divide-se em 2 entregas.

Deverá ser submetido um só ficheiro contendo o código do seu projecto (com o nome LGGG em que L é A - Alameda ou T - Tagus, e GGG é o número do grupo (usar 3 caracteres). O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo. Não é necessário incluir os ficheiros disponibilizados pelo corpo docente.

As entregas têm que ser feitas até ao limite definido a seguir, data e hora, não sendo aceites projectos fora de prazo sob pretexto algum.

- 1^a Entrega - até até às 23:59 do dia 25/10/2019
- 2^a Entrega - até até às 23:59 do dia 06/12/2019

3.1 Condições de realização e discussão dos projectos

Projectos muito semelhantes serão considerados cópia e rejeitados. A detecção de semelhanças entre projectos será realizada utilizando software especializado e caberá exclusivamente ao corpo docente a decisão do que considera ou não cópia. Em caso de cópia, todos os alunos envolvidos terão 0 no projecto, serão reprovados na cadeira e referenciados para o conselho pedagógico.

Os trabalhos serão realizados em grupos de 2 pessoas mas cada pessoa deverá ser capaz de explicar todo o trabalho.

Alguns alunos serão chamados, de forma aleatória ou caso seja necessário confirmar a aquisição a competências, **individualmente** para uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa.

Exemplos

Input:

```
tinittotal = time.process_time()

print("\n(2 val) Exercise 1 - One agent, No limits")
print("Init [30] Goal [56]")
SP = SearchProblem(goal = [56], model = U, auxheur=coords)
tinit = time.process_time()
I = [30]
nn = SP.search(I,limitexp = 2000)
tend = time.process_time()
print("%.1fms"%((tend-tinit)*1000))
if validatepath(nn,I,U):
```

```
        print("path")
        print(nn)
        plotpath(nn,coords)
else:
        print("invalid path")

print("\n(4 val) Exercise 2 - One agent, Limits")
print("Init [30] Goal [56]")
SP = SearchProblem(goal = [56], model = U, auxheur=coords)
tinit = time.process_time()
I = [30]
nn = SP.search(I,limitexp = 2000, tickets = [5,5,2])
tend = time.process_time()
print("%.1fms"%((tend-tinit)*1000))
if validatepath(nn,I,U, tickets = [5,5,2]):
        print("path")
        print(nn)
        plotpath(nn,coords)
else:
        print("invalid path")

print("\n(6 val) Exercise 3 - Three agents, No limits (test 1)")
print("Init [1,3,7] Goal [2,21,9]")
SP = SearchProblem(goal = [2,21,9], model = U, auxheur=coords)
tinit = time.process_time()
I = [1,3,7]
nn = SP.search(I,limitexp = 2000)
tend = time.process_time()
print("%.1fms"%((tend-tinit)*1000))
if validatepath(nn,I,U):
        print("path")
        print(nn)
        plotpath(nn,coords)
else:
        print("invalid path")

print("\n(6 val) Exercise 3 - Three agents, No limits (test 2)")
print("Init [30,40,109] Goal [61,60,71]")
SP = SearchProblem(goal = [61,60,71], model = U, auxheur=coords)
tinit = time.process_time()
I = [30,40,109]
nn = SP.search(I,limitexp = 2000)
tend = time.process_time()
print("%.1fms"%((tend-tinit)*1000))
if validatepath(nn,I,U):
        print("path")
        print(nn)
        plotpath(nn,coords)
else:
        print("invalid path")

print("\n(4 val) Exercise 4 - Three agents, Limits")
print("Init [30,40,109] Goal [61,60,71]")
SP = SearchProblem(goal = [63,61,70], model = U, auxheur=coords)
tinit = time.process_time()
I = [30,40,109]
nn = SP.search(I,limitexp = 3000, limitdepth = 10, tickets = [5,20,2])
tend = time.process_time()
print("%.1fms"%((tend-tinit)*1000))
```

```
if validatepath(nn,I,U, tickets = [5,20,2]):
        print("path")
        print(nn)
        plotpath(nn,coords)
else:
        print("invalid path")

print("\n(4 val) Exercise 5 - Three agents, Limits, Any-Order")

tendtotal = time.process_time()
print("Total time %.1fms"%((tendtotal-tinittotal)*1000))
```

Output:

```
(2 val) Exercise 1 - One agent, No limits
Init [30] Goal [56]
0.0ms
path
[[[], [30]], [[2], [60]], [[2], [72]], [[2], [56]]]

(4 val) Exercise 2 - One agent, Limits
Init [30] Goal [56]
0.0ms
path
[[[], [30]], [[2], [60]], [[2], [72]], [[1], [55]], [[0], [56]]]

(6 val) Exercise 3 - Three agents, No limits (test 1)
Init [1,3,7] Goal [2,21,9]
859.4ms
path
[[[], [1, 3, 7]], [[0, 0, 0], [2, 8, 21]], [[0, 0, 0], [3, 7, 20]], [[0, 0, 0], [2, 21, 9]]]

?

(6 val) Exercise 3 - Three agents, No limits (test 2)
Init [30,40,109] Goal [61,60,71]
765.6ms
path
[[[], [30, 40, 109]], [[2, 0, 2], [60, 53, 72]], [[0, 0, 0], [61, 42, 71]],
 [[0, 0, 0], [62, 52, 82]], [[0, 0, 0], [61, 60, 71]]]

(4 val) Exercise 4 - Three agents, Limits
Init [30,40,109] Goal [61,60,71]
1000.0ms
path
[[[], [30, 40, 109]], [[2, 0, 1], [69, 41, 84]], [[0, 0, 1], [63, 42, 61]],
 [[0, 1, 1], [69, 60, 84]], [[1, 1, 0], [63, 61, 70]]]

(4 val) Exercise 5 - Three agents, Limits, Any-Order
Total time 5562.5ms
```