Inteligência Artificial

IST @ 2018/2019

28 de Setembro de 2018

1. Introdução

Este projecto tem como objectivo desenvolver um programa em Python que resolva diferentes puzzles de uma variante do jogo Solitaire.

2. Descrição da variante do jogo Solitaire

Este é um jogo para um único agente, cuja origem remonta ao século dezassete na corte francesa de Luís XIV. É jogado movimentando berlindes num tabuleiro com pequenos buracos (ou, alternativamente, pinos que encaixam em pequenos buracos).

Figura 1: Exemplo de um tabuleiro da versão francesa do jogo Solitaire[1].

[1] Imagem publicada por Annielogue em http://en.wikipedia.org/wiki/File:French_solitaire.jpg.

Na sua versão mais comum são colocados berlindes em todos os buracos menos no buraco que ocupa a posição central do tabuleiro. O objectivo é, fazendo movimentos válidos, remover todos os berlindes do tabuleiro deixando apenas um na posição central.

Um movimento válido corresponde a fazer saltar um berlinde de forma ortogonal sobre um berlinde adjacente para um buraco que esteja a duas posições do original removendo-se o berlinde sobre o qual foi realizado o salto. Por exemplo:

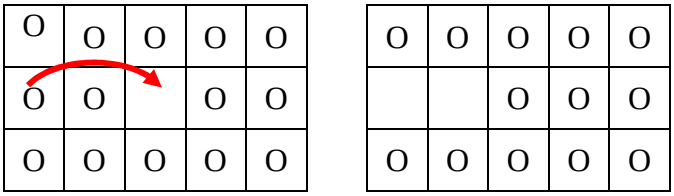

Figura 2: Antes e depois da deslocação para a direita.

Neste trabalho, o jogo difere do jogo original em dois aspectos: (1) o tabuleiro a considerar é um tabuleiro de NxM em que é possível ter mais do que um buraco vazio, ver exemplo na Figura 3; (2) o objectivo é encontrar a sequência de jogadas que deixa apenas 1 berlinde no tabuleiro, sendo que se só sobrar um berlinde este não tem que ficar no centro do tabuleiro.

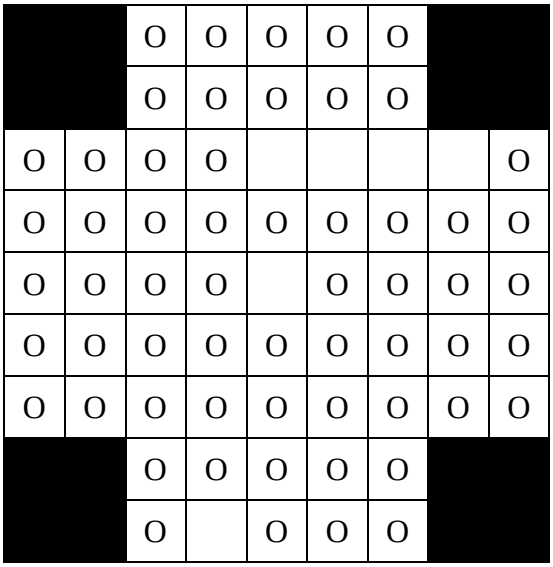

Figura 3: Exemplo de problema a resolver.

3. Resolução do problema e exemplos

O agente implementado deverá ser capaz de lidar com tabuleiros de quaisquer dimensões e com um número variável de espaços em branco.

O objectivo é encontrar uma sequência de jogadas que permitam retirar o número de berlindes necessário para cumprir os requisitos definidos na secção anterior.

Como o tipo de solução a desenvolver pode depender da complexidade dos problemas a resolver, junto se publica um conjunto de problemas tipo:

- Tabuleiro de 5x5 (linhas x colunas)

```
[["_","O","O","O","_"],
 ["O","_","O","_","O"],
 ["_","O","_","O","_"],
 ["O","_","O","_","_"],
 ["_","O","_","_","_"]]
```

- Tabuleiro de 4x4 (linhas x colunas)

```
[["O","O","O","X"],
 ["O","O","O","O"],
 ["O","_","O","O"],
 ["O","O","O","O"]]
```

- Tabuleiro de 4x5 (linhas x colunas)

```
[["O","O","O","X","X"],
 ["O","O","O","O","O"],
 ["O","_","O","_","O"],
 ["O","O","O","O","O"]]
```

- Tabuleiro de 4x6 (linhas x colunas)

```
[["O","O","O","X","X","X"],
 ["O","_","O","O","O","O"],
 ["O","O","O","O","O","O"],
 ["O","O","O","O","O","O"]]
```

4. Implementação

Deve implementar as várias peças de software descritas nesta secção.

4.1. Código Python a utilizar

Para a realização deste projecto devem ser utilizados os algoritmos de procura, desenvolvidos em Python disponibilizados no *site* da cadeira. Este código é uma cópia do código disponibilizado no

site do livro da cadeira, “Artificial Inteligence a Modern Approach”, o qual contém a implementação de vários algoritmos de procura. O mais importante é compreender para que servem, não perceber exactamente como é que as funções estão definidas. Este ficheiro não deve ser alterado, se houver necessidade de alterar definições incluídas neste ficheiro estas devem ser feitas no ficheiro de código desenvolvido que contém a implementação realizada pelo grupo.

4.2. Tipos a desenvolver

Tipo `content`

O tipo `content` representa o conteúdo de uma posição do tabuleiro. Uma posição pode ter uma peça, estar desocupada ou estar bloqueada (não é uma posição válida do tabuleiro do jogo), (1) se a posição está desocupada o conteúdo é “_”, (2) se a posição está ocupada por uma peça o conteúdo é “O” e (3) se está bloqueada o conteúdo é “X”. Note-se que tanto “O” como “X” são letras maiúsculas. A implementação a seguir deve ser usada não devendo ser alterada a representação interna do tipo.

```
# TAI content
def c_peg ():
    return "O"
def c_empty ():
    return "_"
def c_blocked ():
    return "X"
def is_empty (e):
    return e == c_empty()
def is_peg (e):
    return e == c_peg()
def is_blocked (e):
    return e == c_blocked()
```

Tipo `pos`

O tipo `pos` representa as coordenadas de uma posição do tabuleiro. A sua representação interna é um tuplo `(<linha>, <coluna>)`, em que o primeiro elemento corresponde ao índice da linha e o segundo elemento ao índice da coluna. A implementação a seguir deve ser usada não devendo ser alterada a representação interna do tipo.

```
# TAI pos
# Tuplo (l, c)
def  make_pos (l, c):
    return (l, c)
def pos_l (pos):
    return pos[0]
def pos_c (pos):
    return pos[1]
```

Tipo `move`

O tipo `move` representa uma jogada do jogo Solitaire. A sua representação interna é uma lista com dois elementos do tipo `pos`, `[<pos initial>, <pos final>]`, em que o primeiro elemento corresponde à posição inicial da peça a deslocar e o segundo elemento à posição final da peça a deslocar. A implementação a seguir deve ser usada não devendo ser alterada a representação interna do tipo.

```
# TAI move
# Lista [p_initial, p_final]
def make_move (i, f):
    return [i, f]
def move_initial (move):
    return move[0]
def move_final (move):
    return move[1]
```

Tipo `board`

O tipo `board` representa o tabuleiro de um jogo de Solitaire. A representação interna de um jogo de Solitaire é uma lista de listas em que as sublistas correspondem às linhas do tabuleiro do jogo. Uma sublista tem uma representação do conteúdo da linha. O conteúdo de uma posição é representado pelo tipo `content`. O canto superior esquerdo corresponde à posição `(0,0)`. O canto inferior direito corresponde à posição `(<nºlinhas>-1,<nºcolunas>-1)`.

Para o exemplo da figura 3 a representação externa seria:

```
[["X","X","O","O","O","O","O","X","X"],
 ["X","X","O","O","O","O","O","X","X"],
 ["O","O","O","O","O","O","O","O","O"],
 ["O","O","O","O","O","O","O","O","O"],
 ["O","O","O","O","_","O","O","O","O"],
 ["O","O","O","O","O","O","O","O","O"],
 ["O","O","O","O","O","O","O","O","O"],
 ["X","X","O","O","O","O","O","X","X"],
 ["X","X","O","O","O","O","O","X","X"]]
```

Devem ser desenvolvidas pelo menos as seguintes operações:

1. [4 valores] `board_moves(<board>)`que devolve uma lista com todos os movimentos possíveis no tabuleiro.

```
>>> b1 = [["_","O","O","O","_"], ["O","_","O","_","O"], ["_","O","_","O","_"],
          ["O","_","O","_","_"], ["_","O","_","_","_"]]

>>> board_moves(b1)

[[(0, 2), (0, 0)], [(0, 2), (0, 4)], [(0, 2), (2, 2)]]
```

2. [2 valores] `board_perform_move(<board>, <move>)` move a peça da posição inicial para a posição destino removendo a peça que se encontrava entre estas duas posições. Esta operação não deve alterar o tabuleiro que lhe é passado, ou seja, deve começar por criar uma cópia do tabuleiro sobre o qual vai trabalhar e que devolverá no final.

```
>>> board_perform_move(b1, [(0, 2), (0, 0)])

[['O','_','_','O','_'], ['O','_','O','_','O'], ['_','O','_','O','_'],
 ['O','_','O','_','_'], ['_','O','_','_','_']]

>>> b1

[["_","O","O","O","_"], ["O","_","O","_","O"], ["_","O","_","O","_"],
 ["O","_","O","_","_"], ["_","O","_","_","_"]]
```

[2 valores] Tipo `sol_state`

Um estado deve ser representado por uma classe com pelo menos um *slot* chamado `board` em que é armazenada a configuração do tabuleiro a que o estado corresponde. Podem ser acrescentados outros *slots* que sejam considerados úteis.

Para a procura A* e outras procuras informadas é necessário implementar o método:

```
def __lt__(self, <other sol_state>):

    …
```

A necessidade deste método não está relacionada com as estratégias de procura informadas, mas sim com esta implementação específica da gestão da ordenação da lista de abertos.

4.3. [8 valores] Modelação do problema de procura

Deve ser desenvolvida a classe que modela o problema de Solitaire de acordo com a modelação utilizada pelo código de procura referido na secção 4.1. Esta modelação corresponde à definição de uma classe `solitaire` que herda da classe `Problem` e que implementa todos os métodos necessários.

Deve ter em conta os seguintes requisitos:

- A criação de uma instância desta classe deve receber como único argumento um tabuleiro na representação definida acima.

- Uma acção deve ser uma jogada representada pelo tipo `move`, na representação definida acima.

A seguir apresenta-se um protótipo desta classe:

```
class solitaire(Problem):

    """Models a Solitaire problem as a satisfaction problem.
       A solution cannot have more than 1 peg left on the board."""

    def __init__(self, board):
        …

    def actions(self, state):
        …

    def result(self, state, action):
        …

    def goal_test(self, state):
        …

    def path_cost(self, c, state1, action, state2):
        …

    def h(self, node):
        """Needed for informed search."""
        …
```

4.4. Testes

Deve garantir que com o código por si desenvolvido é possível correr a procura em profundidade primeiro, a procura gananciosa e o A* para qualquer problema fornecido. Para obter alguns destes valores pode usar no código publicado a classe `InstrumentedProblem` e o exemplo da sua utilização que se encontra no fim do ficheiro *search.py*.

Para além disso, para os problemas fornecidos na secção 3, usando uma procura informada, deve devolver o resultado correcto em menos de 1 minuto.

4.5. [4 valores] Relatório

Deve produzir um relatório com um máximo de duas páginas que liste para cada um dos problemas fornecidos na secção 3 os resultados obtidos com uma procura em profundidade primeiro, uma procura gananciosa e com uma procura A*. Os resultados devem conter o tempo de execução, o número de nós expandidos e o número de nós gerados.

Deve ser feita uma breve análise crítica dos resultados obtidos.

5. Condições de realização

Cada grupo deve ter 2 elementos e deve fazer a sua inscrição no Fénix.

A entrega do código desenvolvido deve ser feita até às 23h59 do dia 26 de Outubro. Dez dias antes do prazo da entrega (isto é, na Quarta-feira, 16 de Outubro), serão publicadas na página da cadeira as instruções necessárias para a submissão electrónica do código. Apenas a partir dessa altura será possível a submissão por via electrónica. Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverão, portanto, verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretendem que seja avaliada. Não serão abertas excepções.

Atenção: Não são aceites entregas fora do prazo!

Parte da avaliação do trabalho desenvolvido vai decorrer automaticamente, pelo que é essencial que a especificação da interface seja seguida rigorosamente.

O compilador Python a usar é a versão 3.6.2 que pode ser obtida a partir do site:

https://www.python.org/downloads/release/python-362/

O conjunto de funções que implementa o jogador deve ser definido num único ficheiro, este deve poder ser compilado sem erros nem avisos.

No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer caracter que não pertença à tabela ASCII, sob pena de falhar todos os testes automáticos. Isto inclui comentários e cadeias de caracteres.

É prática comum a escrita de mensagens para o ecrã, quando se está a implementar e a testar o código. Isto é ainda mais importante quando se estão a testar/comparar os algoritmos. No entanto, não se esqueça de remover/comentar as mensagens escritas no ecrã na versão final do código entregue. Se não o fizer, correrá o risco de os testes automáticos falharem, e consequentemente ter uma má nota no projecto.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos não implica, pois, que esse projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correcto.

Refira-se ainda que os testes a realizar automaticamente impõem alguns limites espaciais e temporais considerados razoáveis: (1) um limite à heap de 256Mbytes e (2) um tempo máximo por problema de 2 minutos num Intel Core i5 a 2,5GHz, ou seja a eficiência temporal e espacial das soluções apresentadas é relevante.

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 e serão reprovados na cadeira.

6. Discussão dos projectos

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

7. Avaliação

A avaliação desta entrega será feita validando que o código produzido produz os resultados esperados (16 valores) e que o relatório responde ao pedido (4 valores).

Esta entrega contribui com 40% da nota total da componente de avaliação correspondente ao projecto.

8. Novidades do projecto

No caso de haver novidades relativas ao projecto, estas serão afixadas na página da cadeira pelo que esta página deve ser visitada diariamente.