TÉCNICO LISBOA

Inteligência Artificial

1ª parte do Projecto

IST @ 2017/2018

29 de Setembro de 2017

1. Introdução

A 1ª parte do projecto tem como objectivo desenvolver um programa em Python que resolva diferentes puzzles de uma variante do jogo Same Game.

2. Descrição da variante do jogo Same Game

Este é um jogo para um único agente, inventado no Japão em 1985 por Kuniaki Moribe. Nele, existe um tabuleiro rectangular de dimensões arbitrárias. Este tabuleiro encontra-se inicialmente preenchido por completo com peças. Cada peça pode ter uma de várias cores diferentes. O objectivo da variante do jogo é retirar todas as peças do tabuleiro. Na figura 1 mostra-se uma representação do tabuleiro no início do jogo[1].

[1] As várias imagens mostradas ao longo deste enunciado foram retiradas do jogo Same GNOME, Copyright @ 2008 Callum McKenzie.

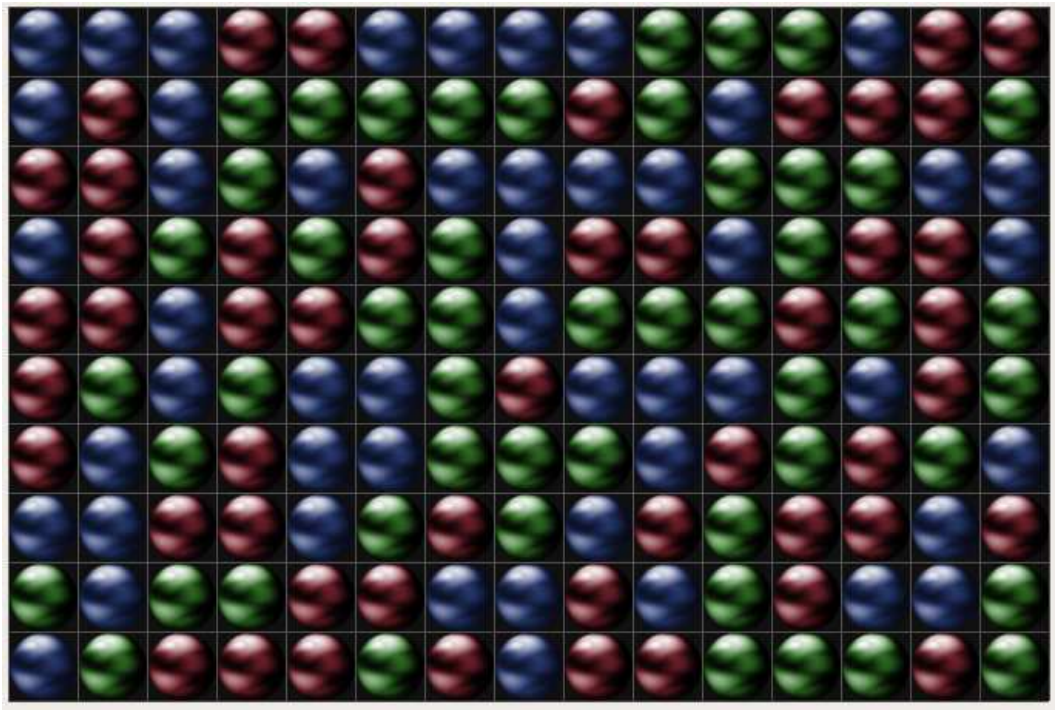
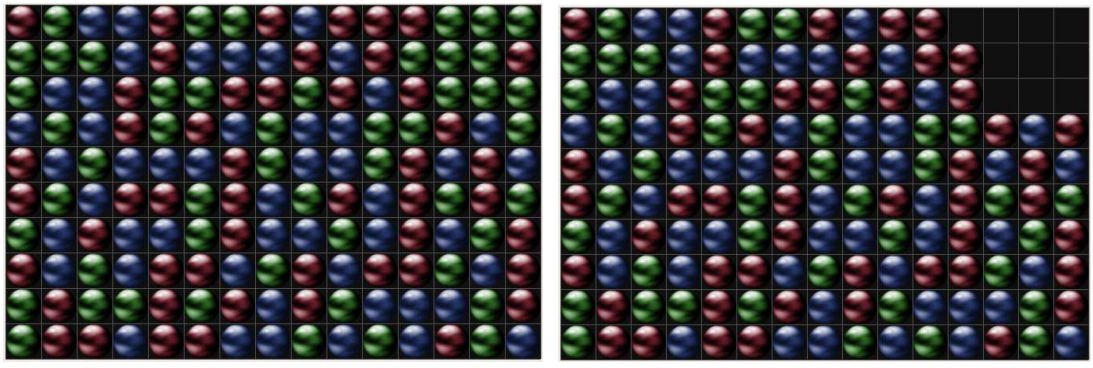


Figura 1: Exemplo de um puzzle do jogo Same Game.

A cada momento, podem ser retirados do tabuleiro conjuntos de pelo menos duas peças adjacentes da mesma cor. Consideram-se adjacentes peças ligadas na horizontal ou vertical, mas não na diagonal. Os conjuntos de peças retirados incluem todas as peças de uma dada cor adjacentes entre si. Não é possível optar por retirar apenas parte de um grupo de peças adjacentes (figura 2).

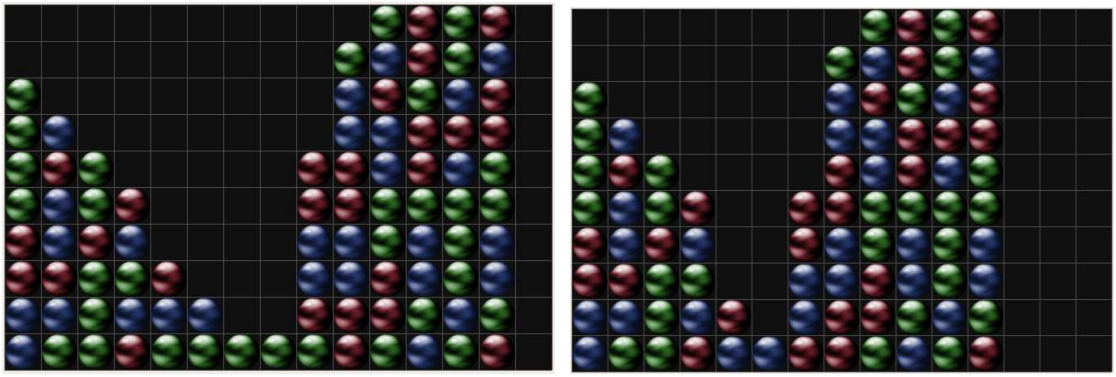


(a) Tabuleiro inicial (b) Tabuleiro após jogada

Figura 2: Exemplo de uma jogada.

Sempre que uma peça é removida, todas as que estão acima dela “caem” de modo a ocupar o espaço por ela deixado. Se, porventura, ao remover uma peça ficarem uma ou mais colunas completamente vazias, então todas as peças à direita dessa coluna movem-se para a esquerda de modo a que não existam zonas desconexas no tabuleiro, nem zonas vazias no lado esquerdo do mesmo (figura 3).

O jogo termina quando o tabuleiro estiver vazio. Um tabuleiro que não esteja vazio e ao qual não seja possível retirar qualquer grupo corresponde a uma derrota do jogador.



(a) Tabuleiro inicial (b) Tabuleiro após jogada

Figura 3: Exemplo de uma jogada.

3. Resolução do problema e exemplos

O agente implementado deverá ser capaz de lidar com tabuleiros de quaisquer dimensões e com qualquer número de cores distintas.

O objectivo é encontrar uma sequência de jogadas que permitam retirar todas as peças do tabuleiro.

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 4x5 (linhas x colunas) com 2 cores sem solução

  `[[1,2,1,2,1],[2,1,2,1,2],[1,2,1,2,1],[2,1,2,1,2]]`

- Tabuleiro de 4x5 (linhas x colunas) com 3 cores

  `[[1,2,2,3,3],[2,2,2,1,3],[1,2,2,2,2],[1,1,1,1,1]]`

- Tabuleiro de 10x4 (linhas x colunas) com 3 cores sem solução

  `[[3,1,3,2],[1,1,1,3],[1,3,2,1],[1,1,3,3],[3,3,1,2],`

```
[2,2,2,2],[3,1,2,3],[2,3,2,3],[5,1,1,3],[4,5,1,2]]
```

- Tabuleiro de 10x4 (linhas x colunas) com 3 cores

```
[[3,1,3,2],[1,1,1,3],[1,3,2,1],[1,1,3,3],[3,3,1,2],
 [2,2,2,2],[3,1,2,3],[2,3,2,3],[2,1,1,3],[2,3,1,2]]
```

- Tabuleiro de 10x4 (linhas x colunas) com 5 cores

```
[[1,1,5,3],[5,3,5,3],[1,2,5,4],[5,2,1,4],[5,3,5,1],
 [5,3,4,4],[5,5,2,5],[1,1,3,1],[1,2,1,3],[3,3,5,5]]
```

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”. Este contém a implementação de vários algoritmos de procura. O mais importante é compreender para que servem e como funcionam. 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 `color`

O tipo cor representa o conteúdo de uma posição do tabuleiro. Uma posição tem uma peça de uma cor ou então está desocupada: (1) se a posição está desocupada, o conteúdo é 0; e (2) se a posição está ocupada por uma peça, o conteúdo é um inteiro maior ou igual a um, correspondente à cor da peça. A implementação a seguir deve ser usada, não devendo ser alterada a representação interna do tipo.

```
# TAI color
# sem cor = 0
# com cor > 0
def get_no_color():
    return 0
def no_color (c):
    return c==0
def color (c):
    return c > 0
```

Tipo pos

O tipo posição 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 group

O tipo group corresponde a uma lista de peças adjacentes com a mesma cor. A sua representação interna é uma lista de elementos do tipo pos.

Tipo board

O tipo tabuleiro representa o tabuleiro de um jogo de Same Game. A representação interna de um jogo de Same Game 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 color. O canto superior esquerdo corresponde à posição (0,0). O canto inferior direito corresponde à posição (<nºlinhas>-1,<nºcolunas>-1).

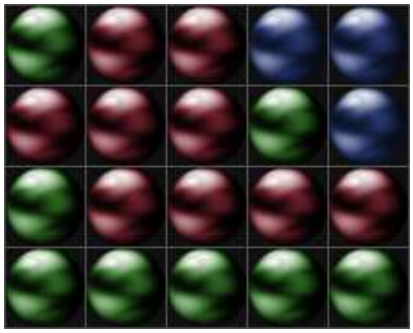


Figura 4: Exemplo de um puzzle de Same Game.

Para o exemplo da figura 4 a representação externa seria:

```
[[1,2,2,3,3],[2,2,2,1,3],[1,2,2,2,2],[1,1,1,1,1]]
```

Devem ser desenvolvidas pelo menos as seguintes operações:

1. [2 valores] `board_find_groups(<board>)` devolve uma lista com os grupos de peças que se podem encontrar no tabuleiro, sendo um grupo de peças uma lista com uma ou mais peças. Na Figura 4, há 5 grupos de peças, dois dos quais só com uma peça.

```
>>> board_find_groups([[1,2,2,3,3],[2,2,2,1,3],[1,2,2,2,2],[1,1,1,1,1]])

[[(0,0)],[(0,1),(1,1),(2,1),(2,2),(1,2),(0,2),(2,3),(2,4),(1,0)],
 [(0,3),(0,4),(1,4)],[(1,3)],[(2,0),(3,0),(3,1),(3,2),(3,3),(3,4)]]
```

2. [5 valores] `board_remove_group(<board>, <group>)` remove o grupo do tabuleiro fazendo a compactação vertical e horizontal das peças. 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.

```
>>> b1 = [[0,0,0,0,0],[0,2,3,3,0],[1,2,1,3,0],[2,2,2,2,0]]

>>>board_remove_group(b1,[(1,1),(2,1),(3,1),(3,2),(3,3),(3,0)])

[[0,0,0,0,0],[0,0,0,0,0],[0,3,3,0,0],[1,1,3,0,0]]
```

Nota: recomenda-se o desenvolvimento de código que permita a escrita de um tabuleiro no ecrã de forma a facilitar a depuração do código.

[1 valor] Tipo `sg_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 sg_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 (a lista de abertos é ordenada por ordem crescente de *f(n)*, este método é usado em caso de empate).

4.3. [8 valores] Modelação do problema de procura

Deve ser desenvolvida a classe que modela o problema de Same Game de acordo com a modelação utilizada pelo código de procura referido na secção 4.1. Esta modelação corresponde à definição de uma classe `same_game` 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 um grupo de peças a remover com uma cardinalidade maior ou igual a dois, na representação definida acima.

Para suportar as procuras informadas, nomeadamente a procura gananciosa e a procura A*, deve desenvolver uma heurística que consiga guiar da forma mais eficiente possível estas procuras. A heurística corresponde à implementação do método `h` da classe `same_game`.

A seguir apresenta-se um protótipo desta classe:

```
class same_game(Problem):

    """Models a Same Game problem as a satisfaction problem.
       A solution cannot have pieces 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

Métodos a testar: a procura em profundidade primeiro, a procura gananciosa e o A*.

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 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 uma procura A*. Os resultados devem conter o tempo de execução, o número de nós expandidos e o número de nós gerados. 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*.

Deve ser feita uma breve análise crítica dos resultados obtidos, em termos de completude, eficiência, comparação entre métodos e heuristica. A análise deve abordar o impacto que a dimensão do tabuleiro e o número de cores tem na procura.

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 20 de Outubro. Duas semanas antes do prazo da entrega (isto é, na Quarta-feira, 6 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

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 5 minutos num Intel Core i5 a 2,5GHz, ou seja a eficiência temporal e espacial das soluções apresentadas é relevante.

Atenção: 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, em grupo ou individual, sendo 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.