Inteligência Artificial 2022/23
Projeto: Bimaru

2 de maio de 2023

1 Introdução

O projeto da unidade curricular de Inteligência Artificial (IA) tem como objetivo desenvolver um programa em Python que resolva o problema Bimaru utilizando técnicas de IA.

O problema Bimaru, também denominado Puzzle Batalha Naval, Yubotu ou Batalha Naval Solitário, é um puzzle inspirado no conhecido jogo de Batalha Naval entre dois jogadores. O jogo foi criado na Argentina por Jaime Poniachik e apareceu pela primeira vez em 1982 na revista argentina *Humor & Juegos*. O jogo ficou conhecido internacionalmente ao ser integrado pela primeira vez no *World Puzzle Championship* em 1992.

2 Descrição do problema

De acordo com a descrição que consta na CSPlib [1], o jogo Bimaru decorre numa grelha quadrada, representando uma área do oceano. Os jogos publicados geralmente usam uma grelha de 10×10, pelo que assumiremos essa dimensão no contexto do projeto.

A área de oceano contém uma frota escondida que o jogador deve encontrar. Esta frota consiste num couraçado (quatro quadrados de comprimento), dois cruzadores (cada um com três quadrados de comprimento), três contratorpedeiros (cada um com dois quadrados de comprimento) e quatro submarinos (um quadrado cada).

Os navios podem ser orientados horizontal ou verticalmente, e dois navios não ocupam quadrados da grelha adjacentes, nem mesmo na diagonal. O jogador também recebe as contagens de linha e coluna, ou seja, o número de quadrados ocupados em cada linha e coluna, e várias dicas. Cada dica especifica o estado de um quadrado individual na grelha: água (o quadrado está vazio); círculo (o quadrado é ocupado por um submarino); meio (este é um quadrado no meio de um couraçado ou cruzador); superior, inferior, esquerda ou direita (este quadrado é a extremidade de um navio que ocupa pelo menos dois quadrados).

A Figura 1 mostra um exemplo da disposição inicial de uma grelha. A Figura 2 mostra uma solução para essa mesma grelha. Podemos assumir que uma instância de Bimaru tem uma **solução única**.

[1] https://www.csplib.org/Problems/prob014/references/

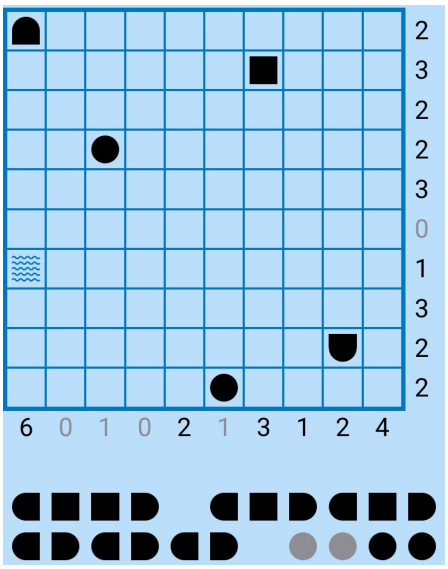

Figura 1: Exemplo de uma instância de Bimaru

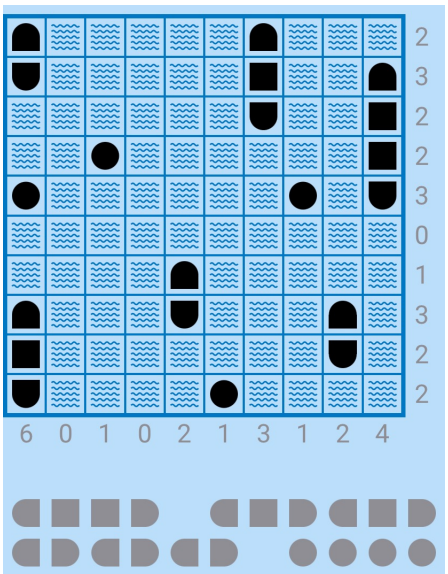

Figura 2: Exemplo de uma solução para uma instância de Bimaru

As imagens que constam no enunciado foram obtidas a partir da aplicação Sea Battle desenvolvida por AculApps para IOS [2] e Android [3].

3 Objetivo

O objetivo deste projeto é o desenvolvimento de um programa em Python 3.8 que, dada uma instância de Bimaru, retorna uma solução, i.e., uma grelha totalmente preenchida.

O programa deve ser desenvolvido num ficheiro `bimaru.py`, que lê uma instância de Bimaru a partir do *standard input* no formato descrito na secção 4.1. O programa deve resolver o problema utilizando uma técnica à escolha e imprimir a solução para o *standard output* no formato descrito na secção 4.2.

Utilização:

```
python3 bimaru.py < <instance_file>
```

4 Formato de input e output

O formato que se segue é baseado no documento *File Format Description for Unsolvable Boards for CSPLib* escrito por Moshe Rubin (Mountain Vista Software) em dezembro de 2005.

4.1 Formato do input

As instâncias do problema Bimaru são constituídas por 3 partes:

1. A primeira linha é iniciada com a palavra `ROW` e contém informação relativa à contagem de posições ocupadas em cada linha da grelha.
2. A segunda linha é iniciada com a palavra `COLUMN` e contém informação relativa à contagem de posições ocupadas em cada coluna da grelha.
3. A terceira linha contém um inteiro que corresponde ao número de dicas.
4. As linhas seguintes são iniciadas com a palavra `HINT` e contêm as dicas correspondentes às posições pré-preenchidas.

Formalmente, cada uma das 4 partes acima descritas tem a seguinte formatação:

1. `ROW <count-0> ... <count-9>`
2. `COLUMN <count-0> ... <count-9>`
3. `<hint total>`

[2] https://apps.apple.com/pt/app/sea-battle-unlimited/id6444275561?l=en
[3] https://play.google.com/store/apps/details?id=ch.aculapps.seabattleunlimited&hl=en

4. `HINT <row> <column> <hint value>`

Os valores possíveis para `<row>` e `<column>` são os números inteiros entre 0 e 9. O canto superior esquerdo da grelha corresponde às coordenadas (0,0).

Os valores possíveis para `<hint value>` são: W (water), C (circle), T (top), M (middle), B (bottom), L (left) e R (right).

Exemplo

O ficheiro de input que descreve a instância da Figura 1 é o seguinte:

```
ROW 2 3 2 2 3 0 1 3 2 2
COLUMN 6 0 1 0 2 1 3 1 2 4
6
HINT 0 0 T
HINT 1 6 M
HINT 3 2 C
HINT 6 0 W
HINT 8 8 B
HINT 9 5 C
```

```
ROW\t2\t3\t2\t2\t3\t0\t1\t3\t2\t2\n
COLUMN\t6\t0\t1\t0\t2\t1\t3\t1\t2\t4\n
6\n
HINT\t0\t0\tT\n
HINT\t1\t6\tM\n
HINT\t3\t2\tC\n
HINT\t6\t0\tW\n
HINT\t8\t8\tB\n
HINT\t9\t5\tC\n
```

4.2 Formato do output

O output do programa deve descrever uma solução para o problema de Bimaru descrito no ficheiro de input, i.e., uma grelha completamente preenchida que respeite as regras previamente enunciadas. O output deve seguir o seguinte formato:

- 10 linhas, onde cada linha indica o conteúdo da respetiva linha da grelha.
- Nas posições pré-preenchidas (correspondentes a dicas) é colocada a respetiva letra maiúscula.
- Nas outras posições são colocadas as respetivas letras, mas minúsculas, com exceção das posições de água que, por questões de legibilidade, são representadas por um ponto.
- Todas as linhas, incluindo a última, são terminadas pelo carater newline, i.e. $\backslash n$

Exemplo

O output que descreve a solução da Figura 2 é:

```
T.....t...
b.....M..t
......b..m
..C......m
c......c.b
..........
W...t.....
t...b...t.
m.......B.
b....C....
```

```
T.....t...\n
b.....M..t\n
......b..m\n
..C......m\n
c......c.b\n
..........\n
W...t.....\n
t...b...t.\n
m.......B.\n
b....C....\n
```

5 Implementação

Nesta secção é descrito o código que poderá ser usado no projeto e o código que deverá ser implementado no projeto.

5.1 Código a utilizar

Para a realização deste projecto devem ser utilizados os ficheiros a ser disponibilizados no site da unidade curricular com a implementação em *Python* dos algoritmos de procura [4]. O mais importante é compreender para que servem e como usar as funcionalidades implementadas nestes ficheiros.

[4]Este código é adaptado a partir do código disponibilizado com o livro *Artificial Intelligence: a Modern Approach* e que está disponível em https://github.com/aimacode.

Estes ficheiros não devem ser alterados. Se houver necessidade de alterar definições incluídas nestes ficheiros, estas alterações devem ser feitas no ficheiro de código desenvolvido que contém a implementação do projeto.

Outras dependências não são permitidas, exceto o python package *numpy*, que pode ser útil para representar a solução e ter acesso a operações sobre arrays.

5.1.1 Procuras

No ficheiro `search.py` estão implementadas as estruturas necessárias para correr os diferentes algoritmos de procura. Destacam-se:

- Classe `Problem`: Representação abstrata do problema de procura;
- Função `breadth_first_tree_search`: Procura em largura primeiro;
- Função `depth_first_tree_search`: Procura em profundidade primeiro;
- Função `greedy_search`: Procura gananciosa;
- Função `astar_search`: Procura A*.

5.1.2 Classe `BimaruState`

Esta classe representa os estados utilizados nos algoritmos de procura. O membro `board` armazena a configuração da grelha a que o estado corresponde. Abaixo é apresentado o código desta classe. Podem ser feitas alterações a esta classe, como por exemplo modificações ao método `__lt__(self, other)` para suportar funções de desempate mais complexas. No entanto, estas alterações devem ser devidamente justificadas com comentários no código.

```
class BimaruState:
    state_id = 0

    def __init__(self, board):
        self.board = board
        self.id = BimaruState.state_id
        BimaruState.state_id += 1

    def __lt__(self, other):
        """ Este método é utilizado em caso de empate na gestão da lista
        de abertos nas procuras informadas. """
        return self.id < other.id
```

5.2 Código a implementar

5.2.1 Classe `Board`

A classe `Board` é a representação interna de uma grelha de Bimaru. A implementação desta classe e respectivos métodos é **livre**. Pode, a título de exemplo, incluir os métodos para determinar valores adjacentes `adjacent_vertical_values` e `adjacent_horizontal_values` que recebem dois argumentos, as coordenadas na grelha (linha, coluna), e devolvem um tuplo com duas strings que correspondem aos valores adjacentes na vertical (acima, abaixo) e na horizontal (esquerda, direita), respectivamente. Caso não existam valores adjacentes, i.e. nas extremidades da grelha, ou caso não tenham ainda sido preenchidos, devolvem `None`. Pode também implementar outros métodos, como por exemplo um método `get_value` que retorne o valor preenchido numa determinada posição, ou um método `print` que imprime a grelha no formato descrito na secção 4.2. Estes métodos poderão ser utilizados para fazer testes à restante implementação da classe.

```
class Board:
    """ Representação interna de uma grelha de Bimaru. """

    def adjacent_vertical_values(self, row: int, col: int) -> (str, str):
        """ Devolve os valores imediatamente acima e abaixo,
        respectivamente. """
        # TODO
        pass

    def adjacent_horizontal_values(self, row: int, col: int) -> (str, str):
        """ Devolve os valores imediatamente à esquerda e à direita,
        respectivamente. """
        # TODO
        pass

    # TODO: outros metodos da classe
```

5.2.2 Função `parse_instance`

A função `parse_instance` é responsável por ler uma instância do problema no formato de input apresentado (secção 4.1) e devolver um objeto do tipo `Board` que a represente. Esta função deve ler a instância a partir do standard input (stdin).

```
    @staticmethod
    def parse_instance():
        """Lê a instância do problema do standard input (stdin)
        e retorna uma instância da classe Board.

        Por exemplo:
            $ python3 bimaru.py < input_T01

            > from sys import stdin
            > line = stdin.readline().split()
        """
        # TODO
        pass
```

5.2.3 Classe `Bimaru`

A classe `Bimaru` herda da classe `Problem` definida no ficheiro `search.py` do código a utilizar e deve implementar os métodos necessários ao seu funcionamento.

O método `actions` recebe como argumento um estado e retorna uma lista de ações que podem ser executadas a partir desse estado. O método `result` recebe como argumento um estado e uma ação, e retorna o resultado de aplicar essa ação a esse estado. Numa primeira abordagem, pode considerar que uma ação corresponde a preencher uma posição livre da grelha. Neste caso, cada ação pode ser representada por um tuplo com 3 valores (índice da linha, índice da coluna, valor a preencher na dada posição). Por exemplo, `(0, 1, w)` representa a ação "preencher a posição da linha 0 e coluna 1 com água". No entando, outras modelações poderão ser mais eficientes, dependendo do problema. Para além de uma modelação baseada em posições individuais da grelha, **sugere-se** considerar, por exemplo, modelações baseadas em navios[5].

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 `Bimaru`. Esta função recebe como argumento um `node`, a partir do qual se pode aceder ao estado atual em `node.state`.

De seguida é disponibilizado um protótipo da classe `Bimaru` que pode ser usado como base para a sua implementação.

[5] Meuffels, W. J. M., & den Hertog, D. (2010). Puzzle—Solving the Battleship puzzle as an integer programming problem. Informs Transactions on Education, 10(3), 156-162, https://pubsonline.informs.org/doi/abs/10.1287/ited.1100.0047.

```python
class Bimaru(Problem):
    def __init__(self, board: Board):
        """ O construtor especifica o estado inicial. """
        # TODO
        pass

    def actions(self, state: BimaruState):
        """ Retorna uma lista de ações que podem ser executadas a
        partir do estado passado como argumento. """
        # TODO
        pass

    def result(self, state: BimaruState, action):
        """ Retorna o estado resultante de executar a 'action' sobre
        'state' passado como argumento. A ação a executar deve ser uma
        das presentes na lista obtida pela execução de
        self.actions(state). """
        # TODO
        pass

    def goal_test(self, state: BimaruState):
        """ Retorna True se e só se o estado passado como argumento é
        um estado objetivo. Deve verificar se todas as posições da grelha
        estão preenchidas de acordo com as regras do problema. """
        # TODO
        pass

    def h(self, node: Node):
        """ Função heuristica utilizada para a procura A*. """
        # TODO
        pass
```

5.2.4 Exemplos de utilização

De seguida, são apresentados alguns exemplos da utilização do código a desenvolver, assim como o respetivo output. Estes exemplos podem ser utilizados para testar a implementação. Considere que o ficheiro `i1.txt` se encontra na diretoria a partir da qual o código está a ser executado e que contém a instância descrita na secção 4.1.

Exemplo 1:

```
# Ler a instância a partir do ficheiro 'i1.txt' (Figura 1):
# $ python3 bimaru.py < i1.txt
board = Board.parse_instance()

# Imprimir valores adjacentes
print(board.adjacent_vertical_values(3, 3))
print(board.adjacent_horizontal_values(3, 3))

print(board.adjacent_vertical_values(1, 0))
print(board.adjacent_horizontal_values(1, 0))
```

Output:

```
(None, None)
(C, None)
(T, None)
(None, None)
```

Exemplo 2:

```
# Ler grelha do ficheiro 'i1.txt' (Figura 1):
# $ python3 bimaru.py < i1.txt
board = Board.parse_instance()

# Criar uma instância de Bimaru:
problem = Bimaru(board)

# Criar um estado com a configuração inicial:
initial_state = BimaruState(board)

# Mostrar valor na posição (3, 3):
print(initial_state.board.get_value(3, 3))

# Realizar acção de inserir o valor w (água) na posição da linha 3 e coluna 3
result_state = problem.result(initial_state, (3, 3, w))

# Mostrar valor na posição (3, 3):
print(result_state.board.get_value(3, 3))
```

Output:

```
None
w
```

Exemplo 3:

```
# Ler grelha do ficheiro 'i1.txt' (Figura 1):
# $ python3 bimaru.py < i1.txt
board = Board.parse_instance()

# Criar uma instância de Bimaru:
problem = Bimaru(board)

# Criar um estado com a configuração inicial:
s0 = BimaruState(board)

# Aplicar as ações que resolvem a instância
s1 = problem.result(s0, (0, 1, w))
s2 = problem.result(s1, (1, 1, w))
s3 = problem.result(s2, (1, 0, b))
s4 = problem.result(s3, (2, 0, w))
s5 = problem.result(s4, (2, 1, w))
# ...
# não estão aqui apresentadas todas as ações
# considere que s94 contém a solução final

# Verificar se foi atingida a solução
print("Is goal?", problem.goal_test(s5))
print("Is goal?", problem.goal_test(s94))
print("Solution:\n", s94.board.print(), sep="")
```

Output:

```
Is goal? False
Is goal? True
Solution:
T.....t...
b.....M..t
......b..m
..C......m
c......c.b
..........
W...t.....
t...b...t.
m.......B.
b....C....
```

Exemplo 4:

```
# Ler grelha do ficheiro 'i1.txt' (Figura 1):
# $ python3 bimaru.py < i1.txt
board = Board.parse_instance()

# Criar uma instância de Bimaru:
problem = Bimaru(board)

# Obter o nó solução usando a procura em profundidade:
goal_node = depth_first_tree_search(problem)

# Verificar se foi atingida a solução
print("Is goal?", problem.goal_test(goal_node.state))
print("Solution:\n", goal_node.state.board.print(), sep="")
```

Output:

```
Is goal? True
Solution:
T.....t...
b.....M..t
......b..m
..C......m
c......c.b
..........
W...t.....
t...b...t.
m.......B.
b....C....
```

O valor de retorno das funções de procura é um objeto do tipo `Node`. Do nó de retorno podem ser retiradas as diversas informaçãoes, por exemplo, estado final (`goal_node.state`), a acção que levou ao estado final `goal_node.action`, e o nó precedente `goal_node.parent`.

6 Avaliação

A nota do projecto será baseada nos seguintes critérios:

- Execução correcta (75% - 15 val.). Estes valores correspondem a testes realizados via submissão no Mooshak.
- Relatório (25% - 5 val.). O relatório será realizado em formato vídeo.

7 Condições de realização e prazos

- O projecto deve ser realizado em grupos de 2 alunos.
- Publicação do enunciado: até ao dia 5 de Maio
- Inscrições de grupos no Fénix: 17 de Maio, até às 17:00
- Entrega do projeto (.py) no Mooshak e relatório: 5 de Junho, até às 17:00

As inscrições dos grupos para o projeto serão feitas através do Fénix; este passo é essencial para posterior acesso ao Mooshak. O código do projeto e relatório têm de ser entregues obrigatoriamente por via electrónica. O código do projeto será entregue através do sistema Mooshak e o relatório em local a designar.

7.1 Mooshak

A avaliação da execução do código do projecto será feita automaticamente através do sistema Mooshak [6]. Após o prazo de inscrição no Fénix e quando notificado pelo corpo docente siga as seguintes instruções para registar e submeter no Mooskak:

- **A partir do dia 19 de maio**, as credênciais de acesso ao Mooshak poderão ser obtidas no seguinte URL utilizando o número de grupo:
  `http://acm.tecnico.ulisboa.pt/~mooshak/cgi-bin/ia2223p4getpass`.
  A senha ser-lhe-á enviada para o email que tem configurado no Fénix. A senha pode não chegar de imediato; aguarde.
- Após ter recebido a sua senha por email, deve efetuar o login no sistema através da página:
  `http://acp.tecnico.ulisboa.pt/~mooshak/`.
  Preencha os campos com a informação fornecida no email.
- Deverá ser submetido o ficheiro *bimaru.py* contendo o código do seu projecto. O ficheiro de código deve conter em comentário, nas primeiras linhas, o número e o nome dos alunos.
- Utilize o botão "Choose file", selecione o ficheiro com extensão .py contendo todo o código do seu projeto. O seu ficheiro .py deve conter a implementação das funções pedidas no enunciado. De seguida clique no botão "Submit" para efetuar a submissão. Aguarde para que o sistema processe a sua submissão!
- Quando a submissão tiver sido processada, poderá visualizar na tabela o resultado correspondente.

Submeta o seu projeto atempadamente, dado que as restrições seguintes podem não lhe permitir fazê-lo no último momento:

[6] A versão de Python utilizada nos testes automáticos é a 3.8.2.

- Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a **última entrega efectuada**.
- Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais.
- O tempo de execução de cada teste está limitado, bem como a memória utilizada.
- Existe um intervalo mínimo entre submissões de **15 minutos**.
- Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

7.2 Relatório em formato de vídeo

Deve produzir um relatório em formato de vídeo com a **duração máxima de 3 minutos**. No vídeo devem aparecer todos os alunos que fazem parte do grupo.

Grupos de alunos que não tenham os meios básicos necessários para a realização do vídeo (i.e. computador com zoom, microfone e câmara) deverão contactar o corpo docente.

8 Cópias

Projectos iguais, ou muito semelhantes, originarão a reprovação na disciplina e, eventualmente, o levantamento de um processo disciplinar. Os programas entregues serão testados em relação a soluções existentes na web. As analogias encontradas com os programas da web serão tratadas como cópias.