# Inteligência Artificial 2021/22 (P4)
# Projeto: Takuzu

9 de maio de 2022

## 1 Introdução

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

## 2 Descrição do problema

O problema Takuzu, também denominado Binairo, Puzzle Binário, Sudoku Binário ou Tic-Tac-Logic, é um jogo lógico que tem duas possíveis origens, ambas em 2009: Tohu wa Vohu foi inventado pelo italiano Adolfo Zanellati e Binairo foi criado pelos belgas Peter De Schepper e Frank Coussement.

O jogo Takuzu decorre sobre um tabuleiro com uma **grelha quadrada**. Cada célula da grelha pode conter os números 0 ou 1.

Dada um tabuleiro com uma grelha $N \times N$, parcialmente preenchida com 0s e 1s, o objetivo do Takuzu é preencher toda a grelha com 0s e 1s tal que:

- Há um número igual de 1s e 0s em cada linha e coluna (ou mais um para grelhas de dimensão ímpar)
- Não há mais do que dois números iguais adjacentes (horizontal ou verticalmente) um ao outro.
- Todas as linhas são diferentes.
- Todas as colunas são diferentes.

O conteúdo das posições da grelha inicialmente preenchidas não pode ser alterado.

A Figura 1 mostra um exemplo da disposição inicial de um tabuleiro com uma grelha $4 \times 4$. A Figura 2 mostra uma solução para esse mesmo tabuleiro. Podemos assumir que uma instância de Takuzu tem uma **única solução**.

## 3 Objetivo

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

| | | | |
|---|---|---|---|
| | 1 | | 0 |
| | | 0 | |
| | 0 | | |
| 1 | 1 | | 0 |

Figura 1: Exemplo de uma instância de Takuzu

| | | | |
|---|---|---|---|
| 0 | **1** | 1 | **0** |
| 1 | 0 | **0** | 1 |
| 0 | **0** | 1 | 1 |
| **1** | **1** | 0 | **0** |

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

O programa deve ser desenvolvido num ficheiro `takuzu.py`, que recebe como argumento da linha de comandos o caminho para um ficheiro que contém um instância de Takuzu no formato descrito na secção 4.1. O programa deve resolver o problema utilizando uma técnica de procura à escolha e imprimir a solução para o *standard output* no formato descrito na secção 4.2.

Utilização:

```
python3 takuzu.py <instance_file>
```

# 4 Formato de input e output

Nos ficheiros de input e output, cada linha corresponde ao conteúdo de cada uma das linhas da grelha.

## 4.1 Formato do input

Os ficheiros de input (*.txt*) representam instâncias do problema Takuzu e seguem o seguinte formato:

- A primeira linha tem apenas um inteiro $N$ que indica a dimensão da grelha $N \times N$;
- As $N$ linhas seguintes indicam o conteúdo de cada uma das $N$ linhas da grelha. Uma posição vazia é representada pelo número 2;
- Cada linha contém `\n` a indicar o seu fim e as colunas são separadas por `\t`.

### Exemplo

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

```
4\n
2\t1\t2\t0\n
2\t2\t0\t2\n
2\t0\t2\t2\n
1\t1\t2\t0\n

4
2 1 2 0
2 2 0 2
2 0 2 2
1 1 2 0
```

## 4.2 Formato do output

O output do programa deve descrever uma solução para o problema de Takuzu descrito no ficheiro de input, i.e., uma grelha completamente preenchida com 0s e 1s que respeite as regras previamente enunciadas. O output deve seguir o seguinte formato:

- cada uma das *N* linhas indica o conteúdo de cada uma das *N* linhas da grelha;
- tanto as linhas como as colunas ocorrem ordenadamente, de forma crescente.
- 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:

```
0\t1\t1\t0\n
1\t0\t0\t1\n
0\t0\t1\t1\n
1\t1\t0\t0\n

0 1 1 0
1 0 0 1
0 0 1 1
1 1 0 0
```

# 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 [1]. O mais importante é compreender para que servem e como usar as funcionalidades implementadas nestes ficheiros.

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**, exepto 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 `Takuzu`

Esta classe representa os estados utilizados nos algoritmos de procura. O membro `board` armazena a configuração do tabuleiro a que o estado corresponde. Abaixo é apresentado o código desta classe. Podem ser feitas alterações a esta classe, estas devem ser devidamente justificadas, por exemplo modificações ao método `__lt__(self, other)` para suportar funções de desempate mais complexas.

[1] 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.

```python
class TakuzuState:
    state_id = 0

    def __init__(self, board):
        self.board = board
        self.id = TakuzuState.state_id
        TakuzuState.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 um tabuleiro de Takuzu. A implementação desta classe e respectivos métodos é livre. Deve, no entanto, incluir os métodos `adjacent_vertical_numbers` e `adjacent_horizontal_numbers` que recebem dois argumentos, as coordenadas no tabuleiro (linha, coluna), e devem devolver um tuplo com dois inteiros que correspondem aos valores imediatos na vertical (abaixo, acima) e na horizontal (esquerda, direita), respectivamente. N.b, caso não exista números adjacentes, i.e. nas extremidades do tabuleiro, deve-se retornar `None` nesses casos. Estes métodos serão utilizados para fazer testes à restante implementação da classe.

```python
class Board:
    """ Representação interna de um tabuleiro de Takuzu. """

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

    def adjacent_horizontal_numbers(self, row: int, col: int) -> (int, int):
        """ 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 o tabuleiro descrito num ficheiro de input e devolver um objeto do tipo `Board` que o represente. Esta função deve ler o tabuleiro do standard input (stdin).

```
    @staticmethod
    def parse_instance_from_stdin():
        """Lê o test do standard input (stdin) que é passado como argumento
        e retorna uma instância da classe Board.

        Por exemplo:
            $ python3 takuzu.py < input_T01

            > from sys import stdin
            > stdin.readline()
        """
        # TODO
        pass

    # TODO: outros metodos da classe
```

### 5.2.3 Classe `Takuzu`

A classe `Takuzu` 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. Em ambos os métodos, uma ação corresponde a preencher 0 ou 1 numa determinada posição. Cada ação é representada sob a forma de um tuplo com 3 inteiros (indíce da linha, indíce da coluna, número a preencher na dada posição), por exemplo, `(2, 1, 1)` representa a ação "preencher o número 1 na posição linha 2 coluna 1".

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 `Takuzu`. 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 `Takuzu` que pode ser usado como base para a sua implementação.

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

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

    def result(self, state: TakuzuState, 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: TakuzuState):
        """ Retorna True se e só se o estado passado como argumento é
        um estado objetivo. Deve verificar se todas as posições do tabuleiro
        estão preenchidas com uma sequência de números adjacentes. """
        # 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 tabuleiro do ficheiro 'i1.txt' (Figura 1):
# $ python3 takuzu < i1.txt
board = Board.parse_instance_from_stdin()
print("Initial:\n", board, sep="")

# Imprimir valores adjacentes
print(board.adjacent_vertical_numbers(3, 3))
print(board.adjacent_horizontal_numbers(3, 3))

print(board.adjacent_vertical_numbers(1, 1))
print(board.adjacent_horizontal_numbers(1, 1))
```

Output:

```
Initial:
2 1 2 0
2 2 0 2
2 0 2 2
1 1 2 0

(None, 2)
(2, None)
(0, 1)
(2, 0)
```

**Exemplo 2:**

```
# Ler tabuleiro do ficheiro 'i1.txt' (Figura 1):
# $ python3 takuzu < i1.txt
board = Board.parse_instance_from_stdin()
print("Initial:\n", board, sep="")

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

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

# Mostrar valor na posição (2, 2):
print(initial_state.board.get_number(2, 2))

# Realizar acção de inserir o número 1 na posição linha 2 e coluna 2
result_state = problem.result(initial_state, (2, 2, 1))

# Mostrar valor na posição (2, 2):
print(result_state.board.get_number(2, 2))
```

Output:

```
Initial:
2 1 2 0
2 2 0 2
2 0 2 2
1 1 2 0

2
1
```

**Exemplo 3:**

```
# Ler tabuleiro do ficheiro 'i1.txt' (Figura 1):
# $ python3 takuzu < i1.txt
board = Board.parse_instance_from_stdin()

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

# Criar um estado com a configuração inicial:
s0 = TakuzuState(board)
print("Initial:\n", s0.board, sep="")

# Aplicar as ações que resolvem a instância
s1 = problem.result(s0, (0, 0, 0))
s2 = problem.result(s1, (0, 2, 1))
s3 = problem.result(s2, (1, 0, 1))
s4 = problem.result(s3, (1, 1, 0))
s5 = problem.result(s4, (1, 3, 1))
s6 = problem.result(s5, (2, 0, 0))
s7 = problem.result(s6, (2, 2, 1))
s8 = problem.result(s7, (2, 3, 1))
s9 = problem.result(s8, (3, 2, 0))

# Verificar se foi atingida a solução
print("Is goal?", problem.goal_test(s9))
print("Solution:\n", s9.board, sep="")
```

Output:

```
Initial:
2 1 2 0
2 2 0 2
2 0 2 2
1 1 2 0

Is goal? True
Solution:
0 1 1 0
1 0 0 1
0 0 1 1
1 1 0 0
```

**Exemplo 4:**

```
# Ler tabuleiro do ficheiro 'i1.txt' (Figura 1):
# $ python3 takuzu < i1.txt
board = Board.parse_instance_from_stdin()

# Criar uma instância de Takuzu:
problem = Takuzu(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, sep="")
```

Output:

```
Is goal? True
Solution:
0 1 1 0
1 0 0 1
0 0 1 1
1 1 0 0
```

O valor de retorno das funções de procura é um objeto do tipo `Node`. Do nó de retorno pode 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.).

# 7 Condições de realização e prazos

- O projecto deve ser realizado em grupos de 2 alunos
- Publicação do enunciado: 9 de Maio

- Inscrições de grupos no Fénix: 23 de Maio, até às 17:00
- Entrega do projeto (.py) no Mooshak e relatório (.pdf) no Fénix: 28 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 através do sistema Mooshak e Fénix, respectivamente.

## 7.1 Mooshak

A avaliação da execução do código do projecto será feita automaticamente através do sistema Mooshak [2]. Após o prazo de inscrição no Fénix e quando notificado pelo corpo docente siga as seguintes instruções para registar e submeter no Mooskak:

- 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/ia2122p4getpass`. A senha ser-lhe-á enviada para o email que tem configurado no Fenix. 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 *takuzu.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 "Browse...", 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:

- 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 (**3 segundos**), bem como a memória utilizada (**32768 KB**).

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

- 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.

Testes adicionais locais: *testes-takuzu.zip* contém testes adicionais que podem ser usados para a implementação do projeto.

### 7.2 Relatório

Deve produzir um relatório contendo um máximo de duas páginas de texto com fonte 12pt e espaçamento normal. Para além destas duas páginas, pode acrescentar ao relatório imagens, figuras e tabelas. O relatório deve conter os resultados obtidos executando uma procura em largura primeiro, 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 análise crítica dos resultados obtidos, comparando em termos de completude e eficiência os diferentes métodos testados. Deve também analisar a heurística implementada e eventualmente compará-la com outras heurísticas avaliadas.

## 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.