Inteligência Artificial 2024/25
Projeto: NURUOMINO

2 de maio de 2025

1 Introdução

O projeto da unidade curricular de Inteligência Artificial (IArt) tem como objetivo desenvolver um programa em Python que resolva uma adaptação do problema NURUOMINO[1] utilizando técnicas de procura e resolução de problemas de Inteligência Artificial.

NURUOMINO, é uma combinação de 'nuru' (Japonês: "pintar") and 'omino' (polinómios). Este puzzle apareceu pela primeira vez na revista *Puzzle Communication Nikoli #106* e é também conhecido por LITS, devido aos quatro (de cinco) tetraminos que podem ser usados neste puzzle neste puzzle: **L**, **I**, **T**, **S**. O tetramino **O**, que forma o quadrado 2x2 não pode ser usado.

2 Descrição do problema

O jogo NURUOMINO decorre numa grelha quadrada $(N \times N)$ e está dividida em várias regiões - polinómios - nenhum com menos de 4 células. O puzzle é resolvido ao colocar um tetramino dentro de cada região de forma a que não existam dois tetrominos iguais ortogonalmente adjacentes (considerando rotações e reflexões como iguais), e que as células preenchidas formem um *nurikabe* válido: todas devem estar ortogonalmente ligadas (formando um único polinómio) e não podem conter tetrominos 2×2.

A Figura 1 mostra as peças L, I, T, S que devem ser colocadas na grelha. Rotações e reflexões correspondem à mesma peça. Figura 2 e Figura 3 e ilustram as rotações e reflexões para a peça **L**, respectivamente.

A Figura 4a mostra um exemplo da disposição inicial de uma grelha 6×6. Para resolver o puzzle é necessário colocar cada uma das peças possiveis, de acordo com as regras acima descritas, em cada região. A Figura 4b mostra uma solução para essa mesma grelha[2]. Podemos assumir que uma instância de NURUOMINO tem uma solução única. Para uma breve explicação do jogo, em video, consulte [1].

[1] https://en.wikipedia.org/wiki/LITS

[2] O exemplo no enunciado foi obtido do seguinte website: https://www.puzzle-lits.com/

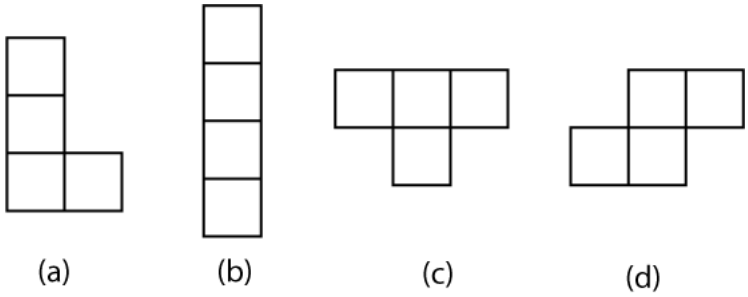


Figura 1: Tetraminos que podem ser usados no puzzle. (a) L (b) I (c) T (d) S

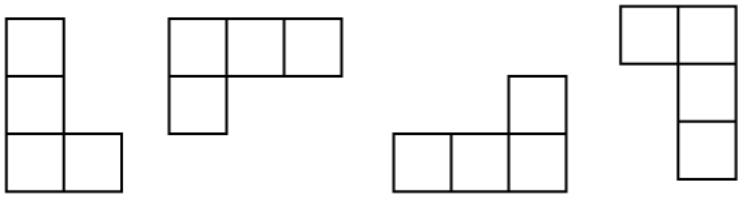


Figura 2: Rotação da peça **L**

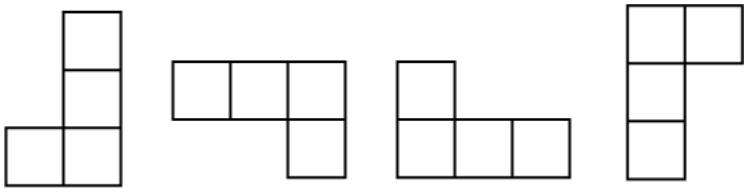


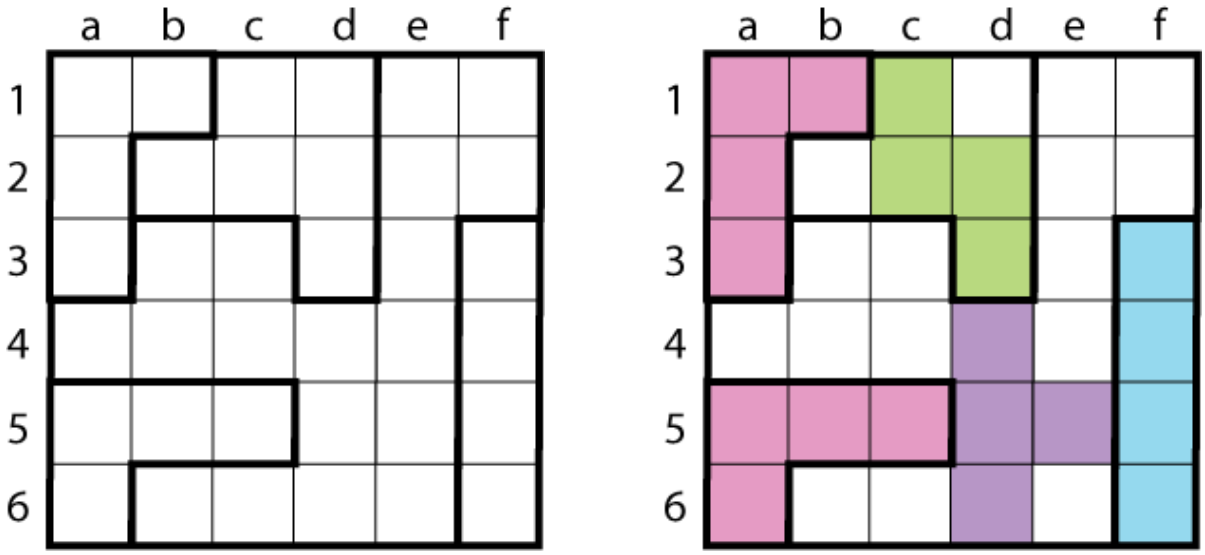
Figura 3: Reflexão da peça **L**

3 Objetivo

O objetivo deste projeto é o desenvolvimento de um programa em Python[3] que, dada uma instância de NURUOMINO, retorna a solução (única), i.e., todas as regiões contêm uma peça de acordo com as regras.

O programa deve ser desenvolvido num ficheiro `nuruomino.py`, que lê uma instância de NURUOMINO 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.

[3]Recomendamos que utilize a versão 3.10, 3.11 ou 3.12



(a) Grelha inicial 6x6 (b) Solução

Figura 4: Exemplo do puzzle antes e depois de resolvido

4 Formato de input e output

Num tabuleiro de NURUOMINO existem K regiões marcadas, como ilustrado na Figura 5. O programa a desenvolver deve ler um ficheiro de texto no formato the input e devolver o formato de output.

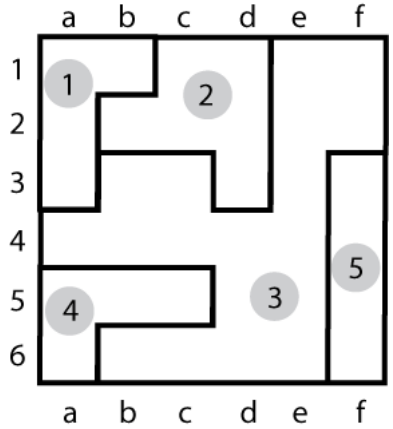


Figura 5: Regiões num tabuleiro numeradas de acordo com o *formato de input*

4.1 Formato do input

As instâncias do problema NURUOMINO seguem o seguinte formato:

```
<region-number-l1c1> ... <region-number-l1cN>
<region-number-l2c1> ... <region-number-l2cN>
...
<region-number-lNc1> ... <region-number-lNcN>
```

Os valores possíveis para `<region-number-*>` são strings de números, indicando que a posição lNcN está inclusa na região *K*. O número de regiões pode variar entre instâncias.

4.1.1 Exemplo

O ficheiro de input que descreve a instância da Figura 4a é o seguinte, de acordo com a numeração em Figura 5:

```
1 1 2 2 3 3
1 2 2 2 3 3
1 3 3 2 3 5
3 3 3 3 3 5
4 4 4 3 3 5
4 3 3 3 3 5
```

```
1\t1\t2\t2\t3\t3\n
1\t2\t2\t2\t3\t3\n
1\t3\t3\t2\t3\t5\n
3\t3\t3\t3\t3\t5\n
4\t4\t4\t3\t3\t5\n
4\t3\t3\t3\t3\t5\n
```

A leitura através do *standard input* faz-se da seguinte forma:

```
python nuruomino.py < text-N.txt
```

4.2 Formato do output

O output do programa deve descrever uma solução para o problema de NURUOMINO descrito no ficheiro de input, i.e., uma grelha em que cada região tem uma peça respeitando as regras previamente enunciadas.

O output deve seguir o seguinte formato:

- Uma linha por cada linha da grelha.
- Cada linha indica o conteúdo da respetiva linha da grelha.

4.2.1 Exemplo

O output que descreve a solução da Figura 4b é:

```
L L S 2 3 3
L 2 S S 3 3
L 3 3 S 3 I
3 3 3 T 3 I
L L L T T I
L 3 3 T 3 I
```

```
L\tL\tS\t2\t3\t3\n
L\t2\tS\tS\t3\t3\n
L\t3\t3\tS\t3\tI\n
3\t3\t3\tT\t3\tI\n
L\tL\tL\tT\tT\tI\n
L\t3\t3\tT\t3\tI\n
```

5 Implementação

Para a realização deste projeto devem ser utilizados os ficheiros Python, disponibilizados na página da unidade curricular, que implementam os algoritmos de procura que irão ser dados ao longo da época letiva [4]. 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 pelo vosso grupo 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.

5.0.1 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*.

[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.

5.0.2 Classe `NuruominoState`

A classe *NuruominoState* define o objeto que representa o estado do tabuleiro em cada instante. 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 NuruominoState:
    state_id = 0

    def __init__(self, board):
        self.board = board
        self.id = Nuroumino.state_id
        Nuroumino.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
```

5.1 Código a implementar

5.1.1 Classe `Board`

A classe `Board` é a representação interna de uma grelha de NURUOMINO. A implementação desta classe e respectivos métodos é livre.

Pode, a título de exemplo, incluir os métodos para determinar regiões ou valores adjacentes `adjacent_regions` e `adjacent_values`. A primeira função recebe o identificador de uma região e devolve uma lista de inteiros com os identificadores das posições adjacentes. A segunda função recebe dois argumentos, as coordenadas na grelha (linha, coluna), e devolve uma lista com os valores das posições adjacentes, incluindo diagonais.

Pode também implementar outros métodos, como por exemplo um método `get_value` que retorne o valor preenchido numa determinada posição, ou um método `print_instance` que imprime a grelha no formato descrito na secção 4.2.

Estes métodos poderão ser utilizados para fazer testes à restante implementação da classe.

```python
class Board:
    """Representação interna de um tabuleiro do Puzzle Nuruomino."""

    def adjacent_regions(self, region:int) -> list:
        """Devolve uma lista das regiões que fazem fronteira com a região enviada no
        #TODO
        pass

    def adjacent_positions(self, row:int, col:int) -> list:
        """Devolve as posições adjacentes à região, em todas as direções,
        incluindo diagonais."""
        #TODO
        pass

    def adjacent_values(self, row:int, col:int) -> list:
        """Devolve os valores das celulas adjacentes à região,
        em todas as direções, incluindo diagonais."""
        #TODO
        pass

    # TODO: outros metodos da classe
```

5.1.2 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).

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

        Por exemplo:
            $ python3 pipe.py < test-01.txt

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

5.1.3 Classe `Nuruomino`

A classe `Nuruomino` 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.

Pode considerar que uma ação corresponde a preencher uma região no tabuleiro com a letra identificadora de uma peça.

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

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

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

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

5.1.4 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 `test01.txt` se encontra na diretoria "./sample-nuruominoboards/"e que contém a instância descrita na secção 4.1.

Exemplo 1:

```
# Ler grelha do figura 1a:
board = Board.parse_instance()

print(board.adjacent_regions(1))
print(board.adjacent_regions(3))
```

Output:

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

Exemplo 2:

```
# Ler grelha do figura 1a:
board = Board.parse_instance()

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

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

# Mostrar valor na posição (2, 1):
print(initial_state.board.get_value(2, 1))

# Realizar ação de rodar 90° clockwise a peça (2, 2)
result_state = problem.result(initial_state, (1, L))

# Mostrar valor na posição (2, 1):
print(result_state.board.get_value(2, 1))

# Mostrar os valores de posições adjacentes
print(result_state.adjacent_values(2,2)
```

Output:

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

Exemplo 3:

```
# Ler grelha do figura 1a:
board = Board.parse_instance()

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

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

# Aplicar as ações que resolvem a instância
s1 = problem.result(s0, (1, 'L', [[1, 1],[1, 0],[1, 0]])
s2 = problem.result(s1, (2, 'S', [[1, 0], [1, 1],[0, 1]])
s3 = problem.result(s2, (3, 'T', [[1, 0],[1, 1],[1, 0]])
s4 = problem.result(s3, (4, 'L', [[1, 1, 1],[1, 0, 0]])
s5 = problem.result(s4, (5, 'I', [[1],[1],[1],[1]])

# Verificar se foi atingida a solução
print("Is goal?", problem.goal_test(s2))
print("Is goal?", problem.goal_test(s5))
print("Solution:\n", s5.board.print(), sep="")
```

Output:

```
Is goal? False
Is goal? True
Solution:
L L S 2 3 3
L 2 S S 3 3
L 3 3 S 3 I
3 3 3 T 3 I
L L L T T I
L 3 3 T 3 I
```

Exemplo 4:

```
# Ler grelha do figura 1a:
board = Board.parse_instance()

# Criar uma instância de Nuruomino:
problem = Nuruomino(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:
L L S 2 3 3
L 2 S S 3 3
L 3 3 S 3 I
3 3 3 T 3 I
L L L T T I
L 3 3 T 3 I
```

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 ação 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 2 de Maio
- Inscrições de grupos no Fénix: até às 17:00 de dia 9 de Maio
- Entrega do projeto (.py) no GitLab interno: 6 de Junho, até às 17:00
- Entrega do relatório no Microsoft Teams: 6 de Junho, até às 23:59

As inscrições dos grupos para o projeto serão feitas através do Fénix; este passo é essencial para posterior acesso ao Gitlab da RNL. O código do projeto e relatório têm de ser entregues obrigatoriamente por via electrónica. O código do projeto será entregue através da plataforma *gitlab.rnl.tecnico.ulisboa.pt* e o relatório através do Microsoft Teams.

7.1 Planeamento

Sugere-se que cada grupo planeie atempadamente a realização do projeto, por exemplo tendo em conta as seguintes metas:

- **Até ao dia 9 de Maio**: Leitura de input / instância de exemplo.
- **Até ao dia 16 de Maio**: Modelação do problema e definição do que é uma ação no contexto do projeto.
- **Até ao dia 23 de Maio**: Implementação das funções de geração de ações e resultados de aplicar uma ação a um estado.
- **Até ao dia 26 de Maio**: Experimentar diferentes procuras e obter soluções corretas para instâncias iniciais (validar formato de output).

7.2 GitLab.rnl

A avaliação da execução do código do projecto será feita automaticamente através da plataforma GitLab.rnl Após o prazo de inscrição no Fénix e quando notificado pelo corpo docente siga as seguintes instruções para registar e submeter no Gitlab:

- **A partir do dia 12 de Maio**, será possível acederem ao Gitlab da RNL para submissão do projeto.
- Para conseguirem aceder a *gitlab.rnl.tecnico.ulisboa.pt* devem garantir em *ciist.ist.utl.pt* que o serviço *afs* está *enabled*. As credenciais de acesso são as mesmas do fénix.
- Deverá ser submetido o ficheiro *nuruomino.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. **É obrigatório**.
- Quando a submissão tiver sido processada, poderá visualizar 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, bem como a memória utilizada.

7.3 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.

O vídeo deve ser estruturado, claro e perceptível, e deve fazer uso de material de apoio ilustrativo que não tem de ser sofisticado (por exemplo: slides, imagens, esquemas, demos). É obrigatório incluir, no final do vídeo, um slide com um disclaimer que explique de forma transparente como foram utilizadas ferramentas de Inteligência Artificial Generativa na realização do projeto. Devem também descrever as heurísticas desenvolvidas, qual o racional e o impacto no desempenho.

Seguem-se algumas sugestões para o conteúdo do vídeo que podem ser usadas se parecerem adequadas:

- Breve descrição do problema (regras e objetivo).
- Descrição da ideia geral para abordar o problema.
- Identificação do que é uma ação no contexto do problema.
- Descrição do que é o resultado de aplicar uma ação num estado.
- Caso exista algum tipo de pre-processamento ou métodos de inferência, estes devem ser mencionados.
- Pequena avaliação experimental comparando diferentes procuras com as instâncias disponibilizadas.
- Identificação da procura final selecionada, quais as suas características (heurística, por exemplo), e motivo da escolha.
- Mostrar como é que a abordagem usada está ligada aos conceitos expostos ao longo da cadeira, de forma critica.

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 Discussões

O corpo docente reserva-se o direito de convocar qualquer grupo para uma discussão oral sobre o trabalho desenvolvido, se considerar necessário, para efeitos de esclarecimento e validação

da autoria.

9 Cópias

Projectos iguais, ou muito semelhantes, originarão a reprovação na disciplina e, eventualmente, o levantamento de um processo disciplinar. Para o efeito será usada a ferramenta Moss da Universidade de Stanford [5].

Os programas entregues serão também testados em relação a soluções existentes na web. As analogias encontradas com os programas da web, incluindo soluções produzidas por *Generative AI*, serão tratadas como cópias.

Referências

[1] A brief explanation of lits. https://youtu.be/q3mspJmENnQ?si=JrczxUu1gDOBiV4Q. Accessed: 2025-05-02.

[5] https://theory.stanford.edu/~aiken/moss/