

# Orbito- $n$

O segundo projeto de Fundamentos da Programação consiste em escrever um programa em Python que permita jogar a uma adaptação do jogo Orbito<sup>1</sup>. Para este efeito, deverá definir um conjunto de tipos abstratos de dados que deverão ser utilizados para manipular a informação necessária no decorrer do jogo, bem como um conjunto de funções adicionais.

## 1 Descrição do jogo

O Orbito é um jogo comercial de tabuleiro abstrato para dois jogadores. Trata-se dum caso particular de jogo  $m, n, k$  com  $m = n = k = 4$ , onde as posições formam duas órbitas. Os jogadores em turnos alternados podem movimentar uma pedra do jogador contrário e a seguir colocar uma pedra própria numa posição livre. No fim de cada turno, todas as pedras rodam uma posição em sentido anti-horário nas suas órbitas. O primeiro jogador que obtiver no fim de um turno  $k = 4$  pedras seguidas da sua cor, horizontalmente, verticalmente ou diagonalmente, é o vencedor. Neste projeto desenvolverá uma versão adaptada do jogo com  $n$  órbitas e sem movimentação de pedras do contrário.

### 1.1 Tabuleiro, posições e pedras

O **tabuleiro** de Orbito- $n$  é uma estrutura quadrangular formado por  $2 \leq n \leq 5$  órbitas. Cada **posição** dum tabuleiro é identificada pela coluna (letra minúscula) e a linha (número) que ocupa. Duas posições são ditas **adjacentes** se estiverem na horizontal, vertical ou diagonal uma da outra, sem outras posições entre elas, e **adjacentes ortogonais** se estiverem na horizontal ou na vertical. Uma posição pode estar livre ou ocupada pela **pedra** de um dos jogadores (branca ou preta, dependendo do jogador). A Figura 1 mostra vários exemplos de tabuleiros de Orbito- $n$ . A **ordem de leitura** das posições dum tabuleiro é de menor órbita (mais interior) a maior órbita (mais exterior), e em caso de empate, ~~da esquerda para a direita seguida de cima para baixo~~ de cima para baixo seguido da esquerda para a direita.

### 1.2 Regras do jogo

Para a realização deste projeto, consideraremos as seguintes regras adaptadas do jogo:

1. **Início:** No início do jogo, o tabuleiro está vazio. O jogador com pedras pretas é o primeiro a jogar. A seguir, os jogadores alternam em turnos subsequentes com o objetivo de obter  $k = 2 \times n$  pedras seguidas próprias.

---

<sup>1</sup><https://flexiqgames.com/en/product/orbito/>

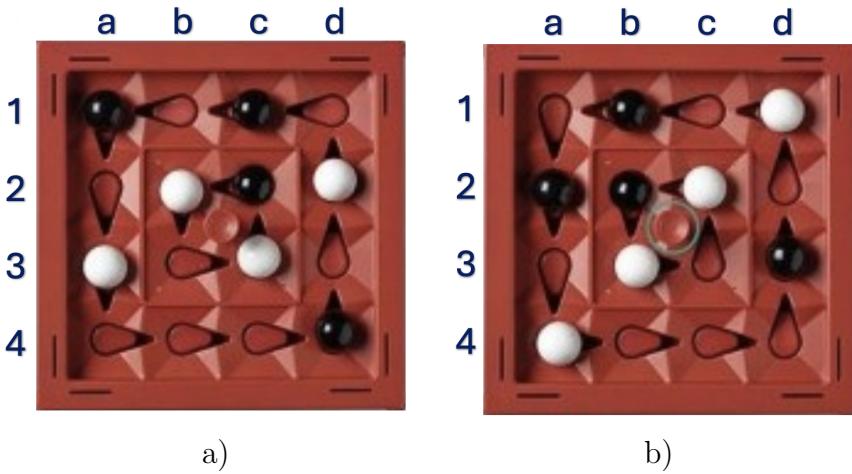


Figura 1: a) Tabuleiro de Orbito- $n$  ( $n = 2$ ) com pedras brancas e pretas. b) Resultado de rodar uma posição todas as pedras do tabuleiro a). A ordem de leitura das posições do tabuleiro é b2, c2, b3, c3, a1, b1, c1, d1, a2, d2, a3, d3, a4, b4, c4, d4.

2. **Turno:** Um turno de um jogador consiste nas seguintes ações (realizadas em ordem):
  - (a) *Colocar:* Coloca uma pedra da sua cor numa posição vazia.
  - (b) *Rodar:* Roda todas as pedras do tabuleiro uma posição sobre a sua órbita em sentido anti-horário.
3. **Fim:** O jogo termina se não existem posições livres no tabuleiro ou, se ao finalizar o turno de um dos jogadores, existe uma linha com  $k$  pedras seguidas iguais, horizontalmente, verticalmente ou diagonalmente.
4. **Vencedor** O jogador com  $k$  pedras próprias seguidas é o vencedor. Se os dois jogadores têm  $k$  pedras seguidas ou se nenhum jogador as têm, o jogo termina em empate.

### 1.3 Estratégia de jogo automático

O programa a desenvolver escolherá a jogada de acordo com a estratégia selecionada. No caso de existir mais de uma posição que cumpra o critério definido pela estratégia selecionada, a posição escolhida é a primeira posição em ordem de leitura do tabuleiro.

#### Estratégia fácil

A posição para colocar a pedra própria é escolhida da seguinte maneira:

**Se** existir no tabuleiro pelo menos uma posição livre que no fim do turno (após rotação) fique adjacente a uma pedra própria, jogar numa dessas posições;

**Se não**, jogar numa posição livre.

## Estratégia normal

A posição para colocar a pedra própria é escolhida da seguinte maneira:

**Determinar** o maior valor de  $L \leq k$  tal que o próprio jogador conseguir colocar  $L$  peças consecutivas que contenha essa jogada no fim do turno atual, ou seja, após uma rotação; ou que o conseguir o adversário no fim do seu seguinte turno, ou seja, após duas rotações. Para esse valor:

**Se** existir pelo menos uma posição que permita no fim do turno obter uma linha que contenha essa posição com  $L$  pedras consecutivas próprias, jogar numa dessas posições;

**Se não**, jogar numa posição que impossibilite o adversário no final do seu próximo turno de obter  $L$  pedras consecutivas numa linha que contenha essa posição.

## 2 Trabalho a realizar

Um dos objetivos deste segundo projeto é definir e implementar um conjunto de Tipos Abstratos de Dados (TAD) que deverão ser utilizados para representar a informação necessária, bem como um conjunto de funções adicionais que permitirão executar corretamente o jogo Orbito- $n$ .

### 2.1 Tipos Abstratos de Dados

Atenção:

- Apenas os construtores e as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.
- Os modificadores, e as funções de alto nível que os utilizam, alteram de modo destrutivo o seu argumento.
- As barreiras de abstração devem ser sempre respeitadas.

#### 2.1.1 TAD *posicao* (1,5 valores)

O TAD imutável<sup>2</sup> *posicao* é usado para representar uma posição do tabuleiro de Orbito- $n$ . As operações básicas associadas a este TAD são:

- Construtor **feito**
  - *cria\_posicao*:  $str \times int \mapsto posicao$   
*cria\_posicao*(*col*, *lin*) recebe um carácter e um inteiro correspondentes à coluna *col* e à linha *lin* e devolve a posição correspondente. O construtor verifica

---

<sup>2</sup>A representação interna dos elementos do tipo deve ser imutável e *hashable*.

a validade dos seus argumentos, gerando um `ValueError` com a mensagem '`cria_posicao: argumentos invalidos`' caso os seus argumentos não sejam válidos.

- Seletores
  - *obtem\_pos\_col*:  $posicao \mapsto str$   
 $obtem\_pos\_col(p)$  devolve a coluna *col* da posição *p*.
  - *obtem\_pos\_lin*:  $posicao \mapsto int$   
 $obtem\_pos\_lin(p)$  devolve a linha *lin* da posição *p*.
- Reconhecedor
  - *eh\_posicao*:  $universal \mapsto booleano$   
 $eh\_posicao(arg)$  devolve `True` caso o seu argumento seja um TAD *posicao* e `False` caso contrário.
- Teste
  - *posicoes\_iguais*:  $universal \times universal \mapsto booleano$   
 $posicoes\_iguais(p1, p2)$  devolve `True` apenas se *p1* e *p2* são posições e são iguais, e `False` caso contrário.
- Transformador
  - *posicao\_para\_str*:  $posicao \mapsto str$   
 $posicao\_para\_str(p)$  devolve a cadeia de caracteres que representa o seu argumento, como mostrado nos exemplos.
  - *str\_para\_posicao*:  $str \mapsto posicao$   
 $str\_para\_posicao(s)$  devolve a posição representada pelo seu argumento.

As funções de alto nível associadas a este TAD são:

- *eh\_posicao\_valida*:  $posicao \times inteiro \mapsto booleano$   
 $eh\_posicao\_valida(p, n)$  devolve `True` se *p* é uma posição válida dentro do tabuleiro de Orbito-*n* e `False` caso contrário.
- *obtem\_posicoes\_adjacentes*:  $posicao \times inteiro \times booleano \mapsto tuplo$   
 $obtem\_posicoes\_adjacentes(p, n, d)$  devolve um *tuplo* com as posições do tabuleiro de Orbito-*n* adjacentes à posição *p* se *d* é `True`, ou as posições adjacentes ortogonais se *d* é `False`. As posições do tuplo são ordenadas em sentido horário começando pela posição acima de *p*.
- *ordena\_posicoes*:  $tuplo \times inteiro \mapsto tuplo$   
 $ordena\_posicoes(t, n)$  devolve um *tuplo* de posições com as mesmas posições de *t* ordenadas de acordo com a ordem de leitura do tabuleiro de Orbito-*n*.

Exemplos de interação:

```
>>> p1 = cria_posicao('a', 21)
Traceback (most recent call last): <...>
ValueError: cria_posicao: argumentos invalidos
>>> p1 = cria_posicao('a', 2)
>>> p2 = cria_posicao('b', 3)
>>> posicoes_iguais(p1, p2)
False
>>> posicao_para_str(p2)
'b3'
>>> posicoes_iguais(p1, str_para_posicao('a2'))
True
>>> tuple(posicao_para_str(p) \
    for p in obtem_posicoes_adjacentes(p1, 2, False))
('a1', 'b2', 'a3')
>>> tuple(posicao_para_str(p) \
    for p in obtem_posicoes_adjacentes(p1, 2, True))
('a1', 'b1', 'b2', 'b3', 'a3')
>>> tup = (cria_posicao('a',1), cria_posicao('a',3),
           cria_posicao('b',1), cria_posicao('b',2))
>>> tuple(posicao_para_str(p) for p in ordena_posicoes(tup, 2))
('b2', 'a1', 'b1', 'a3')
```

### 2.1.2 TAD *pedra* (1,5 valores)

O TAD imutável<sup>3</sup> *pedra* é usado para representar as pedras do jogo. As pedras podem pertencer ao jogador branco ('0') ou ao jogador preto ('X'). Por conveniência, é também definido o conceito pedra *neutra*, que é uma pedra que não pertence a nenhum jogador. As operações básicas associadas a este TAD são:

- Construtor

- *cria\_pedra\_branca*: {}  $\mapsto$  *pedra*  
*cria\_pedra\_branca()* devolve uma pedra pertencente ao jogador branco.
- *cria\_pedra\_preta*: {}  $\mapsto$  *pedra*  
*cria\_pedra\_preta()* devolve uma pedra pertencente ao jogador preto.
- *cria\_pedra\_neutra*: {}  $\mapsto$  *pedra*  
*cria\_pedra\_neutra()* devolve uma pedra neutra.

- Reconhecedor

---

<sup>3</sup>A representação interna dos elementos do tipo deve ser imutável e *hashable*.

- *eh\_pedra*: *universal*  $\mapsto$  *booleano*  
*eh\_pedra(arg)* devolve **True** caso o seu argumento seja um TAD *pedra* e **False** caso contrário.
- *eh\_pedra\_branca*: *pedra*  $\mapsto$  *booleano*  
*eh\_pedra\_branca(p)* devolve **True** caso a *pedra p* seja do jogador branco e **False** caso contrário.
- *eh\_pedra\_preta*: *pedra*  $\mapsto$  *booleano*  
*eh\_pedra\_preta(p)* devolve **True** caso a *pedra p* seja do jogador preto e **False** caso contrário.

- Teste **feito**

- *pedras\_iguais*: *universal*  $\times$  *universal*  $\mapsto$  *booleano*  
*pedras\_iguais(p1, p2)* devolve **True** apenas se *p1* e *p2* são pedras e são iguais.

- Transformador

- *pedra\_para\_str*: *pedra*  $\mapsto$  *str*  
*pedra\_para\_str(p)* devolve a cadeia de caracteres que representa o jogador dono da pedra, isto é, '0', 'X' ou ' ' para pedras do jogador branco, preto ou neutra respetivamente.

As funções de alto nível associadas a este TAD são:

- *eh\_pedra\_jogador*: *pedra*  $\mapsto$  *booleano*  
*eh\_pedra\_jogador(p)* devolve **True** caso a pedra *p* seja de um jogador e **False** caso contrário.
- *pedra\_para\_int*: *pedra*  $\mapsto$  *int*  
*pedra\_para\_int(p)* devolve um *inteiro* valor 1, -1 ou 0, dependendo se a pedra é do jogador preto, branco ou neutra, respetivamente.

Exemplos de interação:

```
>>> b = cria_pedra_branca()
>>> eh_pedra(b)
True
>>> p = cria_pedra_preta()
>>> pedras_iguais(b, p)
False
>>> pedra_para_str(b), pedra_para_str(p)
('0', 'X')
>>> eh_pedra_jogador(cria_pedra_neutra())
False
>>> pedra_para_int(b), pedra_para_int(p)
(-1, 1)
```

### 2.1.3 TAD *tabuleiro* (4,0 valores)

O TAD *tabuleiro* é usado para representar um tabuleiro do jogo Orbito-*n* e as pedras dos jogadores que nele são colocadas. As operações básicas associadas a este TAD são:

- Construtor **feito**
  - *cria\_tabuleiro\_vazio*:  $\text{int} \mapsto \text{tabuleiro}$   
*cria\_tabuleiro\_vazio*(*n*) devolve um *tabuleiro* de Orbito com *n* órbitas, sem posições ocupadas. O construtor verifica a validade do argumento, gerando um *ValueError* com a mensagem '*cria\_tabuleiro\_vazio*: argumento inválido' caso os seu argumento não seja válido. Considere que o número mínimo de órbitas de um tabuleiro de Orbito é 2 e o máximo 5.
  - *cria\_tabuleiro*:  $\text{int} \times \text{tuplo} \times \text{tuplo} \mapsto \text{tabuleiro}$   
*cria\_tabuleiro*(*n*, *tp*, *tb*) devolve um *tabuleiro* de Orbito com *n* órbitas, com as posições do tuplo *tp* ocupadas por pedras pretas e as posições do tuplo *tb* ocupadas por pedras brancas. O construtor verifica a validade dos argumentos, gerando um *ValueError* com a mensagem '*cria\_tabuleiro*: argumentos invalidos' caso os seus argumentos não sejam válidos. Considere que o número mínimo de órbitas de um tabuleiro de Orbito é 2 e o máximo 5.
  - *cria\_copia\_tabuleiro*:  $\text{tabuleiro} \mapsto \text{tabuleiro}$   
*cria\_copia\_tabuleiro*(*t*) recebe um *tabuleiro* e devolve uma cópia do *tabuleiro*.
- Seletores
  - *obtem\_numero\_orbitas*:  $\text{tabuleiro} \mapsto \text{int}$   
*obtem\_numero\_orbitas*(*t*) devolve o número de órbitas do tabuleiro *t*.
  - *obtem\_pedra*:  $\text{tabuleiro} \times \text{posicao} \mapsto \text{pedra}$   
*obtem\_pedra*(*t*, *p*) devolve a pedra na posição *p* do tabuleiro *t*. Se a posição não estiver ocupada, devolve uma pedra *neutra*.
  - *obtem\_linha\_horizontal*:  $\text{tabuleiro} \times \text{posicao} \mapsto \text{tuplo}$   
*obtem\_linha\_horizontal*(*t*, *p*) devolve o tuplo formado por tuplos de dois elementos correspondentes à *posicao* e o valor de todas as posições da linha horizontal que passa pela posição *p*, ordenadas de esquerda para a direita.
  - *obtem\_linha\_vertical*:  $\text{tabuleiro} \times \text{posicao} \mapsto \text{tuplo}$   
*obtem\_linha\_vertical*(*t*, *p*) devolve o tuplo formado por tuplos de dois elementos correspondentes à *posicao* e o valor de todas as posições da linha vertical que passa pela posição *p*, ordenadas de cima para a baixo.
  - *obtem\_linhas\_diagonais*:  $\text{tabuleiro} \times \text{posicao} \mapsto \text{tuplo} \times \text{tuplo}$   
*obtem\_linhas\_diagonais*(*t*, *p*) devolve dois tuplos formados cada um deles por tuplos de dois elementos correspondentes à *posicao* e o valor de todas as posições que formam a diagonal (descendente da esquerda para a direita) e antidiagonal (ascendente da esquerda para a direita) que passam pela posição *p*, respetivamente.

- *obtem\_posicoes\_pedra*:  $\text{tabuleiro} \times \text{pedra} \mapsto \text{tuplo}$   
 $\text{obtem\_posicoes\_pedra}(t, j)$  devolve o tuplo formado por todas as posições do tabuleiro ocupadas por pedras  $j$  (brancas, pretas ou neutras), ordenadas em ordem de leitura do tabuleiro.

- Modificadores [feito](#)

- *coloca\_pedra*:  $\text{tabuleiro} \times \text{posicao} \times \text{pedra} \mapsto \text{tabuleiro}$   
 $\text{coloca\_pedra}(t, p, j)$  modifica destrutivamente o tabuleiro  $t$  colocando a pedra  $j$  na posição  $p$ , e devolve o próprio *tabuleiro*.
- *remove\_pedra*:  $\text{tabuleiro} \times \text{posicao} \mapsto \text{tabuleiro}$   
 $\text{remove\_pedra}(t, p)$  modifica destrutivamente o tabuleiro  $t$  removendo a pedra da posição  $p$ , e devolve o próprio *tabuleiro*.

- Reconhecedor [feito](#)

- *eh\_tabuleiro*:  $\text{universal} \mapsto \text{booleano}$   
 $\text{eh\_tabuleiro}(\text{arg})$  devolve **True** caso o seu argumento seja um TAD *tabuleiro* e **False** caso contrário.

- Teste [feito](#)

- *tabuleiros\_iguais*:  $\text{universal} \times \text{universal} \mapsto \text{booleano}$   
 $\text{tabuleiros\_iguais}(t1, t2)$  devolve **True** apenas se  $t1$  e  $t2$  forem *tabuleiros* e forem iguais.

- Transformador [feito](#)

- *tabuleiro\_para\_str*:  $\text{tabuleiro} \mapsto \text{str}$   
 $\text{tabuleiro\_para\_str}(t)$  devolve a cadeia de caracteres que representa o *tabuleiro* como mostrado nos exemplos.

As funções de alto nível associadas a este TAD são:

- *move\_pedra*:  $\text{tabuleiro} \times \text{posicao} \times \text{posicao} \mapsto \text{tabuleiro}$   
 $\text{move\_pedra}(t, p1, p2)$  modifica destrutivamente o tabuleiro  $t$  movendo a pedra da posição  $p1$  para a posição  $p2$ , e devolve o próprio *tabuleiro*.
- *obtem\_posicao\_seguinte*:  $\text{tabuleiro} \times \text{posicao} \times \text{booleano} \mapsto \text{posicao}$   
 $\text{obtem\_posicao\_seguinte}(t, p, s)$  devolve a posição da mesma órbita que  $p$  que se encontra a seguir no tabuleiro  $t$  em sentido horário se  $s$  for **True** ou anti-horário se for **False**.
- *roda\_tabuleiro*:  $\text{tabuleiro} \mapsto \text{tabuleiro}$   
 $\text{roda\_tabuleiro}(t)$  modifica destrutivamente o tabuleiro  $t$  rodando todas as pedras uma posição em sentido anti-horário, e devolve o próprio *tabuleiro*.

- *verifica\_linha\_pedras*:  $tabuleiro \times posicao \times pedra \times int \mapsto booleano$

*verifica\_linha\_pedras(t, p, j, k)* devolve True se existe pelo menos uma linha (horizontal, vertical ou diagonal) que contenha a posição  $p$  com  $k$  ou mais pedras consecutivas do jogador com pedras  $j$ , e False caso contrário.

Exemplos de interação:

```
>>> t = cria_tabuleiro_vazio(12)
Traceback (most recent call last): <...>
ValueError: cria_tabuleiro_vazio: argumento invalido
>>> t = cria_tabuleiro_vazio(2)
>>> p1 = cria_posicao('c',2)
>>> pedra_para_str(obtem_pedra(t, p1))
' '
>>> b, p = cria_pedra_branca(), cria_pedra_preta()
>>> ib = 'c1', 'c2', 'd2', 'd3', 'd4'
>>> ip = 'a3', 'a4', 'b1', 'b3', 'c3'
>>> ib = tuple(str_para_posicao(i) for i in ib)
>>> ip = tuple(str_para_posicao(i) for i in ip)
>>> for i in ib: coloca_pedra(t, i, b)
>>> for i in ip: coloca_pedra(t, i, p)
>>> print(tabuleiro_para_str(t))
    a    b    c    d
01 [ ]-[X]-[0]-[ ]
    |    |    |    |
02 [ ]-[ ]-[0]-[0]
    |    |    |    |
03 [X]-[X]-[X]-[0]
    |    |    |    |
04 [X]-[ ]-[ ]-[0]
>>> linha = obtem_linha_vertical(t, cria_posicao('c',3))
>>> tuple((posicao_para_str(c), pedra_para_str(v)) for c, v in linha)
([('c1', '0'), ('c2', '0'), ('c3', 'X'), ('c4', ' '))]
>>> col = obtem_linha_horizontal(t, cria_posicao('c',3))
>>> tuple((posicao_para_str(c), pedra_para_str(v)) for c, v in col)
([('a3', 'X'), ('b3', 'X'), ('c3', 'X'), ('d3', '0')])
>>> diag, anti = obtem_linhas_diagonais(t, cria_posicao('c',3))
>>> tuple((posicao_para_str(c), pedra_para_str(v)) for c, v in diag)
([('a1', ' '), ('b2', ' '), ('c3', 'X'), ('d4', '0')])
>>> tuple((posicao_para_str(c), pedra_para_str(v)) for c, v in anti)
([('b4', ' '), ('c3', 'X'), ('d2', '0')])
>>> tuple(posicao_para_str(c) for c in obtem_posicoes_pedra(t, p))
('b3', 'c3', 'b1', 'a3', 'a4')
>>> verifica_linha_pedras(t, cria_posicao('d',1), b, 3))
```

```

False
>>> verifica_linha_pedras(t, cria_posicao('d',2), b, 3))
True
>>> t2 = cria_tabuleiro(2, ip, ib)
>>> tabuleiros_iguais(t, t2)
True
>>> print(tabuleiro_para_str(roda_tabuleiro(t)))
  a   b   c   d
01 [X]-[0]-[ ]-[0]
    |   |   |   |
02 [ ]-[0]-[X]-[0]
    |   |   |   |
03 [ ]-[ ]-[X]-[0]
    |   |   |   |
04 [X]-[X]-[ ]-[ ]
>>> tabuleiros_iguais(t, t2)
False

```

## 2.2 Funções adicionais

### 2.2.1 eh\_vencedor: *tabuleiro* × *pedra* ↦ *booleano* (0,5 valores)

*eh\_vencedor(t, j)* é uma função auxiliar que recebe um *tabuleiro* e uma *pedra* de jogador, e devolve `True` se existe uma linha completa do tabuleiro de pedras do jogador ou `False` caso contrário.

```

>>> ib = tuple(str_para_posicao(i) for i in ('c1','c2','d2','d3','d4'))
>>> ip = tuple(str_para_posicao(i) for i in ('a3','a4','b1','b3','c3'))
>>> t = cria_tabuleiro(2, ip, ib)
>>> b, p = cria_pedra_branca(), cria_pedra_preta()
>>> eh_vencedor(t, p), eh_vencedor(t, b)
(False, False)
>>> _ = coloca_pedra(t, cria_posicao('d',1), b)
>>> eh_vencedor(t, p), eh_vencedor(t, b)
(False, True)

```

### 2.2.2 eh\_fim\_jogo: *tabuleiro* ↦ *booleano* (0,5 valores)

*eh\_fim\_jogo(t)* é uma função auxiliar que recebe um *tabuleiro* e devolve `True` se o jogo já terminou ou `False` caso contrário.

```

>>> eh_fim_jogo(cria_tabuleiro_vazio(2))
False
>>> ib = tuple(str_para_posicao(i) for i in ('c1','c2','d2','d3','d4'))
>>> ip = tuple(str_para_posicao(i) for i in ('a3','a4','b1','b3','c3'))
>>> t = cria_tabuleiro(2, ip, ib)
>>> _ = coloca_pedra(t, cria_posicao('d',1), cria_pedra_branca())
>>> eh_fim_jogo(t)
True

```

### 2.2.3 escolhe\_movimento\_manual: *tabuleiro* $\mapsto$ *posicao* (1,0 valores)

*escolhe\_movimento\_manual(t)* é uma função auxiliar que recebe um *tabuleiro* *t* e permite escolher uma posição livre do tabuleiro onde colocar uma pedra. A função não modifica o seu argumento e devolve a posição escolhida. A função deve apresentar as mensagens do exemplo a seguir, repetindo as mensagens até o jogador introduzir a representação externa de uma jogada válida.

```

>>> ib = tuple(str_para_posicao(i) for i in ('c1','c2','d2','d3','d4'))
>>> ip = tuple(str_para_posicao(i) for i in ('a3','a4','b1','b3','c3'))
>>> t = cria_tabuleiro(2, ip, ib)
>>> print(tabuleiro_para_str(t))
    a   b   c   d
01 [ ]-[X]-[0]-[ ]
    |   |   |   |
02 [ ]-[ ]-[0]-[0]
    |   |   |   |
03 [X]-[X]-[X]-[0]
    |   |   |   |
04 [X]-[ ]-[ ]-[0]
>>> move = escolhe_movimento_manual(t)
Escolha uma posicao livre:d1
>>> posicao_para_str(move)
'd1'
>>> print(tabuleiro_para_str(t))
    a   b   c   d
01 [ ]-[X]-[0]-[ ]
    |   |   |   |
02 [ ]-[ ]-[0]-[0]
    |   |   |   |
03 [X]-[X]-[X]-[0]
    |   |   |   |
04 [X]-[ ]-[ ]-[0]
>>> move = escolhe_movimento_manual(t)

```

```

Escolha uma posicao livre:c1
Escolha uma posicao livre:c3
Escolha uma posicao livre:c4
>>> posicao_para_str(move)
'c4'

```

#### 2.2.4 `escolhe_movimento_auto`: *tabuleiro* × *pedra* × *str* ↦ *posicao* (1,5 valores)

`escolhe_movimento_auto(t, j, lvl)` é uma função auxiliar que recebe um *tabuleiro* *t* (em que o jogo não terminou ainda), uma *pedra* *j*, e a cadeia de caracteres *lvl* correspondente à estratégia, e devolve a posição escolhida automaticamente de acordo com a estratégia selecionada para o jogador com pedras *j*. A função não modifica nenhum dos seus argumentos. As estratégias a seguir devem ser as descritas na seção 1.3 e identificadas pelas cadeias de caracteres 'facil' ou 'normal'.

```

>>> ib = tuple(str_para_posicao(i) for i in ('c1','c2','d2','d3','d4'))
>>> ip = tuple(str_para_posicao(i) for i in ('a3','a4','b1','b3','c3'))
>>> t = cria_tabuleiro(2, ip, ib)
>>> print(tabuleiro_para_str(t))
    a   b   c   d
01 [ ]-[X]-[0]-[ ]
    |   |   |   |
02 [ ]-[ ]-[0]-[0]
    |   |   |   |
03 [X]-[X]-[X]-[0]
    |   |   |   |
04 [X]-[ ]-[ ]-[0]

>>> move_p = escolhe_movimento_auto(t, cria_pedra_preta(), 'facil')
>>> move_b = escolhe_movimento_auto(t, cria_pedra_branca(), 'facil')
>>> posicao_para_str(move_p), posicao_para_str(move_b)
('b2', 'b2')

>>> move_p = escolhe_movimento_auto(t, cria_pedra_preta(), 'normal')
>>> move_b = escolhe_movimento_auto(t, cria_pedra_branca(), 'normal')
>>> posicao_para_str(move_p), posicao_para_str(move_b)
('d1', 'c4')

>>> _ = roda_tabuleiro(roda_tabuleiro(t))
>>> print(tabuleiro_para_str(t))
    a   b   c   d
01 [0]-[ ]-[0]-[0]
    |   |   |   |
02 [X]-[X]-[X]-[0]
    |   |   |   |

```

```

03 [ ]-[0]-[ ]-[ ]
    |   |   |   |
04 [ ]-[X]-[X]-[ ]
>>> move_p = escolhe_movimento_auto(t, cria_pedra_preta(), 'normal')
>>> move_b = escolhe_movimento_auto(t, cria_pedra_branca(), 'normal')
>>> posicao_para_str(move_p), posicao_para_str(move_b)
('d3', 'b1')

```

### 2.2.5 orbito: $\text{int} \times \text{str} \times \text{str} \mapsto \text{int}$ (1,5 valores)

*orbito(n, modo, jog)* é a função principal que permite jogar um jogo completo de Orbito-*n*. A função recebe o número de órbitas do tabuleiro, uma cadeia de caracteres que representa o modo de jogo, e a representação externa de uma *pedra* (preta ou branca), e devolve um inteiro identificando o jogador vencedor (1 para preto ou -1 para branco), ou 0 em caso de empate. O jogo começa sempre com o jogador com pedras pretas e se desenvolve até o fim como descrito na seção 1.2. Os modos de jogo possíveis são:

- 'facil': Jogo de um jogador contra o computador que utiliza a estratégia fácil (sec. 1.3). O jogador joga com as pedras com representação externa *jog*. No fim do jogo a função mostra o resultado obtido pelo jogador: VITORIA, DERROTA ou EMPATE.
- 'normal': Jogo de um jogador contra o computador que utiliza a estratégia normal (sec. 1.3). O jogador joga com as pedras com representação externa *jog*. No fim do jogo a função mostra o resultado obtido pelo jogador: VITORIA, DERROTA ou EMPATE.
- '2jogadores': Jogo de dois jogadores. No fim do jogo a função mostra o resultado do jogo: VITORIA DO JOGADOR 'X', VITORIA DO JOGADOR 'O' ou EMPATE.

A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem '`orbito: argumentos invalidos`' caso os seus argumentos não sejam válidos.

#### Exemplo 1

```

>>> orbito(2, 'facil', '0')
Bem-vindo ao ORBITO-2.
Jogo contra o computador (facil).
0 jogador joga com '0'.
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
    |   |   |   |
02 [ ]-[ ]-[ ]-[ ]
    |   |   |   |

```

```

03 [ ]-[ ]-[ ]-[ ]
|   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do computador (facil):
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
|   |   |   |
02 [ ]-[ ]-[ ]-[ ]
|   |   |   |
03 [ ]-[X]-[ ]-[ ]
|   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:d1
    a   b   c   d
01 [ ]-[ ]-[0]-[ ]
|   |   |   |
02 [ ]-[ ]-[ ]-[ ]
|   |   |   |
03 [ ]-[ ]-[X]-[ ]
|   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do computador (facil):
    a   b   c   d
01 [ ]-[0]-[ ]-[ ]
|   |   |   |
02 [ ]-[ ]-[X]-[ ]
|   |   |   |
03 [ ]-[X]-[ ]-[ ]
|   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:c1
    a   b   c   d
01 [0]-[0]-[ ]-[ ]
|   |   |   |
02 [ ]-[X]-[ ]-[ ]
|   |   |   |
03 [ ]-[ ]-[X]-[ ]
|   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do computador (facil):
    a   b   c   d

```

```

01 [0]-[ ]-[ ]-[ ]
| | | |
02 [0]-[X]-[X]-[ ]
| | | |
03 [ ]-[X]-[ ]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:b1
      a   b   c   d
01 [0]-[ ]-[ ]-[ ]
| | | |
02 [0]-[X]-[ ]-[ ]
| | | |
03 [0]-[X]-[X]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[ ]
Turno do computador (facil):
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [0]-[X]-[X]-[ ]
| | | |
03 [0]-[X]-[X]-[ ]
| | | |
04 [0]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:b4
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[X]-[X]-[ ]
| | | |
03 [0]-[X]-[X]-[ ]
| | | |
04 [0]-[0]-[0]-[ ]
Turno do computador (facil):
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [X]-[X]-[X]-[ ]
| | | |
03 [ ]-[X]-[X]-[ ]

```

04	[0]	-[0]	-[0]
VITORIA			
-1			

### Exemplo 2

```
>>> orbito(2, 'normal', 'X')
Bem-vindo ao ORBITO-2.
Jogo contra o computador (normal).
0 jogador joga com 'X'.
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
02 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
03 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:c2
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
02 [ ]-[X]-[ ]-[ ]
      |   |   |   |
03 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do computador (normal):
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
02 [ ]-[0]-[ ]-[ ]
      |   |   |   |
03 [ ]-[X]-[ ]-[ ]
      |   |   |   |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:a3
      a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
      |   |   |   |
```

```

02 [ ]-[ ]-[ ]-[ ]
| | | |
03 [ ]-[0]-[X]-[ ]
| | | |
04 [X]-[ ]-[ ]-[ ]
Turno do computador (normal):
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[ ]-[X]-[ ]
| | | |
03 [ ]-[0]-[0]-[ ]
| | | |
04 [ ]-[X]-[ ]-[ ]
Turno do jogador.
Escolha uma posicao livre:b2
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[X]-[0]-[ ]
| | | |
03 [ ]-[X]-[0]-[ ]
| | | |
04 [ ]-[ ]-[X]-[ ]
Turno do computador (normal):
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [0]-[0]-[0]-[ ]
| | | |
03 [ ]-[X]-[X]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[X]
Turno do jogador.
Escolha uma posicao livre:a2
Escolha uma posicao livre:a3
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[0]-[X]-[ ]
| | | |
03 [0]-[0]-[X]-[X]
| | | |

```

04 [X]-[ ]-[ ]-[ ]

Turno do computador (normal):

a b c d

01 [ ]-[ ]-[ ]-[ ]

| | | |

02 [ ]-[X]-[X]-[X]

| | | |

03 [0]-[0]-[0]-[ ]

| | | |

04 [0]-[X]-[ ]-[ ]

Turno do jogador.

Escolha uma posicao livre:a2

a b c d

01 [ ]-[ ]-[ ]-[X]

| | | |

02 [ ]-[X]-[0]-[ ]

| | | |

03 [X]-[X]-[0]-[ ]

| | | |

04 [0]-[0]-[X]-[ ]

Turno do computador (normal):

a b c d

01 [ ]-[ ]-[X]-[ ]

| | | |

02 [0]-[0]-[0]-[ ]

| | | |

03 [ ]-[X]-[X]-[ ]

| | | |

04 [X]-[0]-[0]-[X]

Turno do jogador.

Escolha uma posicao livre:b1

a b c d

01 [X]-[X]-[ ]-[ ]

| | | |

02 [ ]-[0]-[X]-[ ]

| | | |

03 [0]-[0]-[X]-[X]

| | | |

04 [ ]-[X]-[0]-[0]

Turno do computador (normal):

a b c d

01 [X]-[ ]-[ ]-[ ]

| | | |

```

02 [X]-[X]-[X]-[X]
| | | |
03 [O]-[O]-[O]-[O]
| | | |
04 [O]-[ ]-[X]-[O]
EMPATE
0

```

### Exemplo 3

```

>>> orbito(2, '2jogadores', 'X')
Bem-vindo ao ORBITO-2.
Jogo para dois jogadores.
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[ ]-[ ]-[ ]
| | | |
03 [ ]-[ ]-[ ]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador 'X'.
Escolha uma posicao livre:a1
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [X]-[ ]-[ ]-[ ]
| | | |
03 [ ]-[ ]-[ ]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador 'O'.
Escolha uma posicao livre:b2
    a   b   c   d
01 [ ]-[ ]-[ ]-[ ]
| | | |
02 [ ]-[ ]-[ ]-[ ]
| | | |
03 [X]-[O]-[ ]-[ ]
| | | |
04 [ ]-[ ]-[ ]-[ ]
Turno do jogador 'X'.
Escolha uma posicao livre:a4

```

	a	b	c	d
01	[ ]-[ ]-[ ]-[ ]			
02	[ ]-[ ]-[ ]-[ ]			
03	[ ]-[ ]-[0]-[ ]			
04	[X]-[X]-[ ]-[ ]			

Turno do jogador '0'.

Escolha uma posicao livre:c4

	a	b	c	d
01	[ ]-[ ]-[ ]-[ ]			
02	[ ]-[ ]-[0]-[ ]			
03	[ ]-[ ]-[ ]-[ ]			
04	[ ]-[X]-[X]-[0]			

Turno do jogador 'X'.

Escolha uma posicao livre:a4

	a	b	c	d
01	[ ]-[ ]-[ ]-[ ]			
02	[ ]-[0]-[ ]-[ ]			
03	[ ]-[ ]-[ ]-[0]			
04	[ ]-[X]-[X]-[X]			

Turno do jogador '0'.

Escolha uma posicao livre:b3

	a	b	c	d
01	[ ]-[ ]-[ ]-[ ]			
02	[ ]-[ ]-[ ]-[0]			
03	[ ]-[0]-[0]-[X]			
04	[ ]-[ ]-[X]-[X]			

Turno do jogador 'X'.

Escolha uma posicao livre:b4

	a	b	c	d
01	[ ]-[ ]-[ ]-[0]			

```

02 [ ]-[ ]-[0]-[X]
|   |   |   |
03 [ ]-[ ]-[0]-[X]
|   |   |   |
04 [ ]-[ ]-[X]-[X]
Turno do jogador '0'.
Escolha uma posicao livre:a3
    a    b    c    d
01 [ ]-[ ]-[0]-[X]
|   |   |   |
02 [ ]-[0]-[0]-[X]
|   |   |   |
03 [ ]-[ ]-[ ]-[X]
|   |   |   |
04 [0]-[ ]-[ ]-[X]
VITORIA DO JOGADOR 'X'
1

```

### 3 Condições de Realização e Prazos

- A entrega do 2º projeto será efetuada exclusivamente por via eletrónica. Para submeter o seu projeto deverá realizar pelo menos uma atualização do repositório remoto GitLab fornecido pelo corpo docente, até às **17:00 do dia 28 de Outubro de 2024**. Depois desta hora, qualquer atualização do repositório será ignorada. Não serão aceites submissões de projetos por outras vias sob pretexto algum.
- A solução do projeto deverá consistir apenas num único ficheiro com extensão `.py` contendo todo o código do seu projeto.
- Cada aluno tem direito a **15 submissões sem penalização**. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.
- Será considerada para avaliação a **última** submissão (mesmo que tenha pontuação inferior a submissões anteriores). Deverá, portanto, verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada.
- Submissões que não corram nenhum dos testes automáticos por causa de pequenos erros de sintaxe ou de codificação, poderão ser corrigidos pelo corpo docente, incorrendo numa penalização de três valores.
- Não é permitida a utilização de qualquer módulo ou função não disponível built-in no Python 3, ou seja, não são permitidos `import`, com exceção da função `reduce` do `functools`.

- Pode, ou não, haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).
- Lembre-se que no Técnico, a fraude académica é levada muito a sério e que a cópia numa prova (projetos incluídos) leva à reprovação na disciplina e eventualmente a um processo disciplinar. Os projetos serão submetidos a um sistema automático de deteção de cópias<sup>4</sup>, o corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.
- A submissão do projeto por parte dos alunos, é interpretada pelo corpo docente como uma declaração de honra conforme cada aluno (ou grupo) é o autor único de todo o trabalho apresentado.

## 4 Submissão

A submissão do projeto de FP é realizada atualizando o repositório remoto GitLab privado fornecido pelo corpo docente para cada aluno (ou grupo de projeto). O endereço web do repositório do projeto dos alunos é [https://gitlab.rnl.tecnico.ulisboa.pt/ist-fp/fp24/prj2/\(curso\)/\(grupo\)](https://gitlab.rnl.tecnico.ulisboa.pt/ist-fp/fp24/prj2/(curso)/(grupo)), onde:

- (curso) pode ser leic-a, leic-t, leti ou leme;
- (grupo) pode ser:
  - o ist-id para os alunos da LEIC-A, LEIC-T e LETI (ex. ist190000);
  - g seguido dos dois dígitos que identificam o número de grupo de projeto dos alunos da LEME (ex. g00).

Sempre que é realizada uma nova atualização do repositório remoto é desencadeado o processo de avaliação automática do projeto e é contabilizada uma nova submissão. Quando a submissão tiver sido processada, poderá visualizar um relatório de execução com os detalhes da avaliação automática do seu projeto em [http://fp.rnl.tecnico.ulisboa.pt/fp24p2/reports/\(grupo\)](http://fp.rnl.tecnico.ulisboa.pt/fp24p2/reports/(grupo)). Adicionalmente, receberá no seu email o mesmo relatório. Se não receber o email ou o relatório web aparentar não ter sido atualizado, contacte o corpo docente. Note que o sistema de submissão e avaliação não limita o número de submissões simultâneas. Um número elevado de submissões num determinado momento, poderá ocasionar a rejeição de alguns pedidos de avaliação. Para evitar problemas de último momento, **recomenda-se que submeta o seu projeto atempadamente**.

Detalhes sobre como aceder ao GitLab, configurar o par de chaves SSH, executar os comandos de Git e recomendações sobre ferramentas, encontram-se na página da disciplina na seção “Material de Apoio - Ambiente de Desenvolvimento”<sup>5</sup>.

---

<sup>4</sup><https://theory.stanford.edu/~aiken/moss>

<sup>5</sup><https://fenix.tecnico.ulisboa.pt/disciplinas/FProg11/2024-2025/1-semestre/ambiente-de-desenvolvimento>

## 5 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. **Execução correta (60%).** A avaliação da correta execução será feita com um conjunto de testes unitários utilizando o módulo de Python pytest<sup>6</sup>.

Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. Como os testes de execução valem 60% (equivalente a 12 valores) da nota do projeto, uma submissão obtém a nota máxima de 1200 pontos por esta componente.

O facto de um projeto completar com sucesso os testes públicos fornecidos não implica que esse projeto esteja totalmente correto, pois estes não são exaustivos. É da responsabilidade de cada aluno garantir que o código produzido está de acordo com a especificação do enunciado usando testes próprios adicionais, de forma a completar com sucesso os testes privados.

2. **Respeito pelas barreiras de abstração (20%).** A avaliação do respeito pelas barreiras de abstração também será feita automaticamente utilizando o módulo de Python pytest. Para este fim, serão usados um conjunto de testes (diferentes dos testes de execução) especificamente definidos para testar que o código desenvolvido

pelos alunos respeita as barreiras de abstração. Como os testes de abstração valem 20% (equivalente a 4 valores) da nota do projeto, uma submissão obtém a nota máxima de 400 pontos por esta componente.

3. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura. Em particular, serão consideradas as seguintes componentes:

- Boas práticas (1,5 valores): serão considerados entre outros a clareza do código, elementos de programação funcional, integração de conhecimento adquirido durante a UC, a criatividade das soluções propostas e a escolha da representação adotada nos TADs.
- Comentários (1 valor): deverão incluir a assinatura dos TADs (incluindo representação interna adotada e assinatura das operações básicas), assim como a assinatura de cada função definida, comentários para o utilizador (*docsstring*) e comentários para o programador.
- Tamanho de funções, duplicação de código e abstração procedural (1 valor)
- Escolha de nomes (0,5 valores).

## 6 Recomendações e aspetos a evitar

As seguintes recomendações e aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

---

<sup>6</sup><https://docs.pytest.org/en/7.4.x/>

- Leia todo o enunciado, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.
- No processo de desenvolvimento do projeto, comece por implementar as várias funções pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina.
- Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.
- Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá sentir a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).
- Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
- Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.
- A atitude “vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo” é totalmente errada.
- Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As “marteladas” no código têm o efeito de distorcer cada vez mais o código.