TÉCNICO LISBOA

Fundamentos da Programação
Ano lectivo 2024-25
Primeiro Projecto
4 de Outubro de 2024

MNK

Neste primeiro projeto de Fundamentos da Programação os alunos irão desenvolver as funções de forma a implementar um programa em Python que permita um jogador humano jogar contra o computador num jogo do tipo m, n, k[1].

1 Descrição do jogo

1.1 Regras do jogo

Um jogo m, n, k é um jogo de tabuleiro abstrato em que dois jogadores colocam pedras de forma alternada nas posições livres de um tabuleiro de dimensão $m \times n$. O jogador que obtiver primeiro k pedras seguidas da sua própria cor, horizontalmente, verticalmente ou diagonalmente (diagonal ou antidiagonal), é o vencedor. Trata-se duma generalização de jogos populares, como o Jogo do Galo[2] ($m = n = k = 3$) ou o Gomoku[3] ($m = n = 15, k = 5$). Como as pedras, após colocadas, não são movimentadas nem retiradas do tabuleiro, é habitual jogar em *lápis e papel*, utilizando os símbolos `'X'` e `'O'` em vez de pedras pretas e brancas.

1.2 Estratégia de jogo

Os jogos de tipo m, n, k apresentam uma combinatória simples que permite desenvolver soluções eficientes para escolher o melhor movimento baseadas em algoritmos de teoria de jogos[4]. Alternativamente, neste projeto exploraremos algumas heurísticas simples. Em particular, o programa a desenvolver escolherá a jogada seguinte de acordo com uma das seguintes estratégias:

Estratégia fácil

> **Se** existir no tabuleiro pelo menos uma posição livre e adjacente a uma pedra própria, jogar numa dessas posições;
>
> **Se não**, jogar numa posição livre.

[1] https://en.wikipedia.org/wiki/M,n,k-game
[2] https://en.wikipedia.org/wiki/Tic-tac-toe
[3] https://en.wikipedia.org/wiki/Gomoku
[4] https://en.wikipedia.org/wiki/Minimax

Estratégia normal

Determinar o maior valor de $L \leq k$ tal que o próprio ou o adversário podem conseguir colocar L peças consecutivas na próxima jogada numa linha vertical, horizontal ou diagonal que contenha essa jogada. Para esse valor:

Se existir pelo menos uma posição que permita 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 de obter L pedras consecutivas numa linha que contenha essa posição.

Estratégia difícil se existe uma posição onde podemos ganhar, jogar lá;

Se existir pelo menos uma posição que permita obter uma linha própria com k pedras consecutivas (e ganhar o jogo), jogar numa dessas posições;

Se não, e se existir pelo menos uma posição que impossibilite ao adversário de obter uma linha com k pedras consecutivas (e ganhar o jogo), jogar numa dessas posições;

Se não, para cada posição livre, simular um jogo até ao fim em que o jogador atual joga nessa posição e o resto de jogadas são determinadas assumindo que os dois jogadores alternadamente (o adversário e o próprio) jogam seguindo uma estratégia de jogo *normal*. Registar o resultado de cada simulação e escolher a posição que leva ao melhor resultado possível, isto é:

Se existir pelo menos uma posição/simulação que permitiria ganhar o jogo, jogar numa dessas posições;

Se não, e se existir pelo menos uma posição/simulação que permitiria empatar o jogo, escolher uma dessas posições;

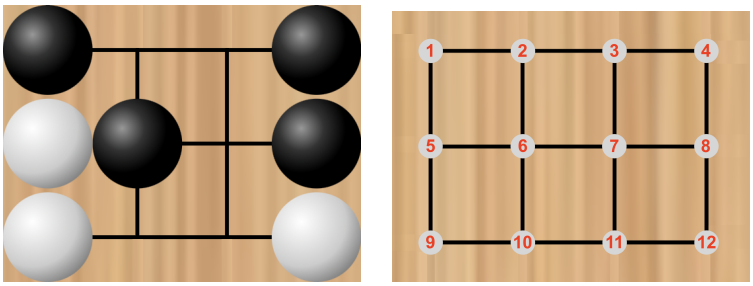
Se não, jogar numa posição livre.

2 Trabalho a realizar

O objetivo do primeiro projeto é escrever um programa em Python, correspondendo às funções descritas nesta secção, que permita jogar um jogo m, n, k conforme descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso seja necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida o devem fazer, para as restantes assume-se que os argumentos estão corretos.

2.1 Representação do tabuleiro

Considere que um *tabuleiro* de dimensão $m \times n$ é representado internamente (ou seja, no seu programa) por um tuplo com m tuplos. Cada um dos m tuplos contém n valores inteiros. Os valores representam cada uma das posições do tabuleiro podendo tomar valores iguais a 1, -1 ou 0, dependendo se a posição estiver ocupada por uma pedra preta, branca ou se estiver livre, respetivamente. Assim, o tabuleiro da Figura 1a) é definido pelo tuplo `((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))`.



a) b)

Figura 1: a) Tabuleiro de tamanho 3×4 com pedras pretas e brancas. b) Posições dum tabuleiro de tamanho 3×4 a vermelho.

Considere também que cada uma das *posições* dum tabuleiro é representada internamente por um número inteiro positivo seguindo a ordem da esquerda para a direita e de cima para baixo como ilustrado na Figura 1b). Assim, as pedras pretas da Figura 1a) ocupam as posições $1, 4, 6, 8$, e as brancas as posições $5, 9, 12$. Duas posições são adjacentes se estiverem na horizontal, vertical ou diagonal uma da outra, sem outras posições entre elas. A *distância* entre duas posições adjacentes é igual a 1. A distância entre duas posições quaisquer é o menor número de posições adjacentes que as conectam. Notar que uma definição alternativa e equivalente de distância entre posições, corresponde à distância de *Chebyshev* ou L_∞[5], interpretando o tabuleiro como um sistema de coordenadas com espaçamento vertical e horizontal entre posições consecutivas igual a 1 unidade.
Por fim, define-se a *posição central* dum tabuleiro, como a posição com valor igual a $c = (m \div 2) \times n + n \div 2 + 1$, sendo $\div$ a divisão inteira. Por exemplo, o centro do tabuleiro da Figura 1b) é a posição 7, enquanto que as posições $2, 3, 4, 6, 8, 10, 11, 12$ estão a distância 1 do centro, e as posições $1, 5, 9$ a distância 2.

2.1.1 eh_tabuleiro: universal → booleano (0,5 valores)

eh_tabuleiro(arg) recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a um tabuleiro e `False` caso contrário, sem nunca gerar erros. Nesta parte do projeto, considere que um tabuleiro corresponde a um tuplo de tuplos como descrito, em que $2 \leq m, n \leq 100$.

[5] https://en.wikipedia.org/wiki/Chebyshev_distance

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> eh_tabuleiro(tab)
True
>>> tab = ((1,0,0,1),(-1,1,'O',1), (-1,0,0,-1))
>>> eh_tabuleiro(tab)
False
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0))
>>> eh_tabuleiro(tab)
False
```

2.1.2 eh_posicao: universal → booleano (0,25 valores)

eh_posicao(arg) recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a uma posição dum tabuleiro e `False` caso contrário, sem nunca gerar erros. Considere que as posições no tabuleiro são definidas por um inteiro, como indicado na Figura 1b).

```
>>> eh_posicao(9)
True
>>> eh_posicao(-2)
False
>>> eh_posicao((1,))
False
```

2.1.3 obtem_dimensao: tabuleiro → tuplo (0,25 valores)

obtem_dimensao(tab) recebe um tabuleiro e devolve um tuplo formado pelo número de linhas m e colunas n do tabuleiro.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_dimensao(tab)
(3, 4)
>>> tab = ((1,0,0),(-1,1,0), (-1,0,1),(1,-1,0),(0,-1,0))
>>> obtem_dimensao(tab)
(5, 3)
```

2.1.4 obtem_valor: tabuleiro × posicao → inteiro (0,5 valores)

obtem_valor(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve o valor contido nessa posição.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_valor(tab, 1)
1
>>> obtem_valor(tab, 2)
0
>>> obtem_valor(tab, 5)
-1
```

2.1.5 obtem_coluna: tabuleiro × posicao → tuplo (0,5 valores)

obtem_coluna(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve um tuplo com todas as posições que formam a coluna em que esta contida a posição, ordenadas de menor a maior.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_coluna(tab, 2)
(2, 6, 10)
>>> obtem_coluna(tab, 7)
(3, 7, 11)
```

2.1.6 obtem_linha: tabuleiro × posicao → tuplo (0,5 valores)

obtem_linha(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve um tuplo com todas as posições que formam a linha em que esta contida a posição, ordenadas de menor a maior.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_linha(tab, 3)
(1, 2, 3, 4)
>>> obtem_linha(tab, 7)
(5, 6, 7, 8)
```

2.1.7 obtem_diagonais: tabuleiro × posicao → tuplo (1,25 valores)

obtem_diagonais(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve o tuplo formado por dois tuplos de posições correspondentes à diagonal (descendente da esquerda para a direita) e antidiagonal (ascendente da esquerda para a direita) que passam pela posição, respetivamente.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_diagonais(tab, 6)
```

```
((1, 6, 11), (9, 6, 3))
>>> obtem_diagonais(tab, 5)
((5, 10), (5, 2))
```

2.1.8 tabuleiro_para_str: tabuleiro → cad. carateres (1,25 valores)

tabuleiro_para_str(tab) recebe um tabuleiro e devolve a cadeia de caracteres que o representa (a representação externa ou representação "para os nossos olhos"), de acordo com o exemplo na seguinte interação.

```
>>> tab = ((1,0,0),(-1,1,0), (-1,0,0))
>>> tabuleiro_para_str(tab)
'X---+---+\n|   |   |\nO---X---+\n|   |   |\nO---+---+'
>>> print(tabuleiro_para_str(tab))
X---+---+
|   |   |
O---X---+
|   |   |
O---+---+
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> print(tabuleiro_para_str(tab))
X---+---+---X
|   |   |   |
O---X---+---X
|   |   |   |
O---+---+---O
```

2.2 Funções de inspeção e manipulação do tabuleiro

2.2.1 eh_posicao_valida: tabuleiro × posicao → booleano (0,25 valores)

eh_posicao_valida(tab, pos) recebe um tabuleiro e uma posição, e devolve `True` se a posição corresponde a uma posição do tabuleiro, e `False` caso contrário. Se algum dos argumentos dados for inválido, a função deve gerar um erro com a mensagem `'eh_posicao_valida: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> eh_posicao_valida(tab, 9)
True
>>> eh_posicao_valida(tab, 12)
True
>>> eh_posicao_valida(tab, 13)
False
```

2.2.2 eh_posicao_livre: tabuleiro × posicao → booleano (0,25 valores)

eh_posicao_livre(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve `True` se a posição corresponde a uma posição livre (não ocupada por pedras), e `False` caso contrário. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'eh_posicao_livre: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> eh_posicao_livre(tab, 2)
True
>>> eh_posicao_livre(tab, 4)
False
>>> eh_posicao_livre(tab, 12)
False
```

2.2.3 obtem_posicoes_livres: tabuleiro → tuplo (0,5 valores)

obtem_posicoes_livres(tab) recebe um tabuleiro e devolve o tuplo com todas as posições livres do tabuleiro, ordenadas de menor a maior. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'obtem_posicoes_livres: argumento invalido'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_posicoes_livres(tab)
(2, 3, 7, 10, 11)
>>> tab = ((1,-1,0),(1,-1,0),(1,-1))
>>> obtem_posicoes_livres(tab)
Traceback (most recent call last): <...>
ValueError: obtem_posicoes_livres: argumento invalido
```

2.2.4 obtem_posicoes_jogador: tabuleiro × inteiro → tuplo (0,5 valores)

obtem_posicoes_jogador(tab, jog) recebe um tabuleiro e um inteiro identificando um jogador (1 para o jogador com pedras pretas ou -1 para o jogador com pedras brancas) e devolve o tuplo com todas as posições do tabuleiro ocupadas por pedras do jogador, ordenadas de menor a maior. Se algum dos argumentos dados for inválidos, a função deve gerar um erro com a mensagem `'obtem_posicoes_jogador: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_posicoes_jogador(tab, 1)
(1, 4, 6, 8)
>>> obtem_posicoes_jogador(tab, -1)
(5, 9, 12)
```

```
>>> obtem_posicoes_jogador(tab, -2)
Traceback (most recent call last): <...>
ValueError: obtem_posicoes_jogador: argumentos invalidos
```

2.2.5 obtem_posicoes_adjacentes: tabuleiro × posicao → tuplo (0,75 valores)

obtem_posicoes_adjacentes(tab, pos) recebe um tabuleiro e uma posição do tabuleiro, e devolve o tuplo formado pelas posições do tabuleiro adjacentes (horizontal, vertical e diagonal), ordenadas de menor a maior. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'obtem_posicoes_adjacentes: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> obtem_posicoes_adjacentes(tab, 1)
(2, 5, 6)
>>> obtem_posicoes_adjacentes(tab, 6)
(1, 2, 3, 5, 7, 9, 10, 11)
>>> obtem_posicoes_adjacentes(tab, 13)
Traceback (most recent call last): <...>
ValueError: obtem_posicoes_adjacentes: argumentos invalidos
```

2.2.6 ordena_posicoes_tabuleiro: tabuleiro × tuplo → tuplo (0,75 valores)

ordena_posicoes_tabuleiro(tab, tup) recebe um tabuleiro e um tuplo de posições do tabuleiro (potencialmente vazio), e devolve o tuplo com as posições em ordem ascendente de distância à posição central do tabuleiro. Posições com igual distância à posição central, são ordenadas de menor a maior de acordo com a posição que ocupam no tabuleiro. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'ordena_posicoes_tabuleiro: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> ordena_posicoes_tabuleiro(tab, tuple(range(1,13)))
(7, 2, 3, 4, 6, 8, 10, 11, 12, 1, 5, 9)
>>> ordena_posicoes_tabuleiro(tab, '123')
Traceback (most recent call last): <...>
ValueError: ordena_posicoes_tabuleiro: argumentos invalidos
```

2.2.7 marca_posicao: tabuleiro × posicao × inteiro → tabuleiro (0,75 valores)

marca_posicao(tab, pos, jog) recebe um tabuleiro, uma posição *livre* do tabuleiro e um inteiro identificando um jogador (1 para o jogador com pedras pretas ou -1 para o

jogador com pedras brancas), e devolve um novo tabuleiro com uma nova pedra do jogador indicado nessa posição. Se algum dos argumentos dados for inválidos, a função deve gerar um erro com a mensagem `'marca_posicao: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> marca_posicao(tab, 11, -1)
((1, 0, 0, 1), (-1, 1, 0, 1), (-1, 0, -1, -1))
>>> marca_posicao(tab, 7, -1)
((1, 0, 0, 1), (-1, 1, -1, 1), (-1, 0, 0, -1))
>>> marca_posicao(tab, 1, -1)
Traceback (most recent call last): <...>
ValueError: marca_posicao: argumentos invalidos
```

2.2.8 verifica_k_linhas: tabuleiro × posicao × inteiro × inteiro → booleano (1,25 valores)

verifica_k_linhas(tab, pos, jog, k) recebe um tabuleiro, uma posição do tabuleiro, um valor inteiro identificando um jogador (1 para o jogador com pedras pretas ou -1 para o jogador com pedras brancas), e um valor inteiro positivo k, e devolve `True` se existe pelo menos uma linha (horizontal, vertical ou diagonal) que contenha a posição com k ou mais pedras consecutivas do jogador indicado , e `False` caso contrário. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'verifica_k_linhas: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> verifica_k_linhas(tab, 4, 1, 2), verifica_k_linhas(tab, 12, 1, 2))
(True, False)
>>> verifica_k_linhas(tab, 1, 1, 3), verifica_k_linhas(tab, 9, -1, 3)
(False, False)
>>> tab = ((1,0,0,0),(-1,1,0,1), (-1,0,1,-1), (0,0,0,0))
>>> verifica_k_linhas(tab, 6, 1, 3), verifica_k_linhas(tab, 16, 1, 3))
(True, False)
>>> verifica_k_linhas(tab, 2, 1, -3)
Traceback (most recent call last): <...>
ValueError: verifica_k_linhas: argumentos invalidos
```

2.3 Funções de jogo

2.3.1 eh_fim_jogo: tabuleiro × inteiro → booleano (0,5 valores)

eh_fim_jogo(tab, k) recebe um tabuleiro e um valor inteiro positivo k, e devolve um booleano a indicar se o jogo terminou (`True`) ou não (`False`). Um jogo pode terminar caso um dos jogadores tenha k pedras consecutivas, ou caso já não existam mais posições

livres para marcar. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'eh_fim_jogo: argumentos invalidos'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> eh_fim_jogo(tab, 3)
False
>>> tab = ((1,0,0,0),(-1,1,0,1), (-1,0,1,-1))
>>> eh_fim_jogo(tab, 3)
True
>>> tab = ((1,1,-1),(-1,-1,1),(1,-1,1))
>>> eh_fim_jogo(tab, 3)
True
>>> eh_fim_jogo(((1,-1),(-1,-1,1)), 2)
Traceback (most recent call last): <...>
ValueError: eh_fim_jogo: argumentos invalidos
```

2.3.2 escolhe_posicao_manual: tabuleiro → posicao (0,5 valores)

escolhe_posicao_manual(tab) recebe um tabuleiro e devolve uma posição introduzida manualmente pelo jogador. A função deve apresentar a mensagem do exemplo a seguir, repetindo a mensagem até o jogador introduzir uma posição livre. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'escolhe_posicao_manual: argumento invalido'`.

```
>>> tab = ((1,0,0,1),(-1,1,0,1), (-1,0,0,-1))
>>> escolhe_posicao_manual(tab)
Turno do jogador. Escolha uma posicao livre: 2
2
>>> escolhe_posicao_manual(tab)
Turno do jogador. Escolha uma posicao livre: 1
Turno do jogador. Escolha uma posicao livre: 13
Turno do jogador. Escolha uma posicao livre: 3
3
>>> escolhe_posicao_manual((1,1))
Traceback (most recent call last): <...>
ValueError: escolhe_posicao_manual: argumento invalido
```

2.3.3 escolhe_posicao_auto: tabuleiro × inteiro × inteiro × cad. carateres → posicao (3,5 valores)

escolhe_posicao_auto(tab, jog, k, lvl) recebe um tabuleiro (em que o jogo não terminou ainda), um inteiro identificando um jogador (1 para o jogador com pedras pretas ou -1 para o jogador com pedras brancas), um inteiro positivo correspondendo ao valor k dum

jogo m, n, k, e a cadeia de carateres correspondente à estratégia, e devolve a posição escolhida automaticamente de acordo com a estratégia selecionada. As estratégias a seguir devem ser as descritas na seção 1.2 e identificadas pelas cadeias de cararateres `'facil'`, `'normal'` ou `'dificil'`. Sempre que houver mais do que uma posição que cumpra um dos critérios definidos nas estratégias anteriores, deve escolher a posição mais próxima da posição central do tabuleiro, como definido nas seções 2.1 e 2.2.6. Se algum dos argumentos dados for inválidos, a função deve gerar um erro com a mensagem `'escolhe_posicao_auto: argumentos invalidos'`.

```
>>> tab = ((0,0,0),(0,1,0),(-1,0,1))
>>> escolhe_posicao_auto(tab, -1, 3, 'facil')
4
>>> escolhe_posicao_auto(tab, -1, 3, 'normal')
1
>>> tab = ((0,0,-1),(-1,1,0),(1,0,0))
>>> escolhe_posicao_auto(tab, 1, 3, 'normal')
1
>>> escolhe_posicao_auto(tab, 1, 3, 'dificil')
8
```

2.3.4 jogo_mnk: tuplo × inteiro × cad. carateres → inteiro (1,5 valores)

jogo_mnk(cfg, jog, lvl) é a função principal que permite jogar um jogo completo m, n, k de um jogador contra o computador. A função recebe um tuplo de três valores inteiros correspondentes aos valores de configuração do jogo m, n e k; um inteiro identificando a cor das pedras do jogador humano (1 para as pedras pretas ou -1 para as pedras brancas); e uma cadeia de caracteres identificando a estratégia de jogo utilizada pela máquina. O jogo começa sempre com o jogador com pedras pretas a marcar uma posição livre e termina quando um dos jogadores vence ou se não existirem posições livres no tabuleiro. A função mostra o resultado do jogo (`VITORIA`, `DERROTA` ou `EMPATE`) e devolve um inteiro identificando o jogador vencedor (1 para preto ou -1 para branco), ou 0 em caso de empate. A função deve verificar a validade dos seus argumentos, gerando um erro com a mensagem `'jogo_mnk: argumentos invalidos'`.

Exemplo 1

```
>>> jogo_mnk((3,3,3), 1, 'facil')
Bem-vindo ao JOGO MNK.
O jogador joga com 'X'.
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
```

```
Turno do jogador. Escolha uma posicao livre: 5
+---+---+
|   |   |
+---X---+
|   |   |
+---+---+
Turno do computador (facil):
O---+---+
|   |   |
+---X---+
|   |   |
+---+---+
Turno do jogador. Escolha uma posicao livre: 4
O---+---+
|   |   |
X---X---+
|   |   |
+---+---+
Turno do computador (facil):
O---O---+
|   |   |
X---X---+
|   |   |
+---+---+
Turno do jogador. Escolha uma posicao livre: 6
O---O---+
|   |   |
X---X---X
|   |   |
+---+---+
VITORIA
1
```

Exemplo 2

```
>>> jogo_mnk((3,3,3), -1, 'dificil')
Bem-vindo ao JOGO MNK.
O jogador joga com 'O'.
+---+---+
|   |   |
+---+---+
|   |   |
+---+---+
Turno do computador (dificil):
```

```
X---+---+
|   |   |
+---+---+
|   |   |
+---+---+
Turno do jogador. Escolha uma posicao livre: 2
X---O---+
|   |   |
+---+---+
|   |   |
+---+---+
Turno do computador (dificil):
X---O---+
|   |   |
X---+---+
|   |   |
+---+---+
Turno do jogador. Escolha uma posicao livre: 7
X---O---+
|   |   |
X---+---+
|   |   |
O---+---+
Turno do computador (dificil):
X---O---+
|   |   |
X---X---+
|   |   |
O---+---+
Turno do jogador. Escolha uma posicao livre: 6
X---O---+
|   |   |
X---X---O
|   |   |
O---+---+
Turno do computador (dificil):
X---O---+
|   |   |
X---X---O
|   |   |
O---+---X
DERROTA
1
```

3 Condições de Realização e Prazos

- A entrega do 1º 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 16 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.

- Pode, ou não, haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

- Lembre-se de que no Técnico, a fraude académica é levada muito a sério e que a cópia em qualquer momento de avaliação (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[6], 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 a 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/prj1/(curso)/(grupo)`, onde:

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

- `(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/fp24p1/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 com 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"[7].

5 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. **Avaliação automática (80%).** A avaliação da correta execução será feita com um conjunto de testes unitários utilizando o módulo de Python pytest[8].
   Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. Como a avaliação automática vale 80% (equivalente a 16 valores) da nota do projeto, uma submissão obtém a nota máxima de 1600 pontos.
   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. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura. Em particular, serão consideradas as seguintes componentes:

[7] `https://fenix.tecnico.ulisboa.pt/disciplinas/FProg11/2024-2025/1-semestre/ambiente-de-desenvolvimento`

[8] `https://docs.pytest.org/en/7.4.x/`

- Boas práticas (1,5 valores): serão considerados entre outros a clareza do código, a integração de conhecimento adquirido durante a UC e a criatividade das soluções propostas.
- Comentários (1 valor): deverão incluir a assinatura das funções definidas, comentários para o utilizador (*docstring*) e comentários para o programador.
- Tamanho de funções, duplicação de código e abstração procedimental (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):

- 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.