TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2022-23
Segundo Projeto
21 de Outubro de 2022

Minas

O segundo projeto de Fundamentos da Programação consiste em escrever um programa em Python que permita jogar ao videojogo minas (*mineswepper*[1]). 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 jogo das minas é um videojogo para um único jogador do tipo lógico, que se tornou extremamente popular após a sua inclusão no sistema operativo Microsoft Windows desde a sua versão 3.1. O jogo apresenta um campo com várias parcelas em que se encontram algumas minas escondidas. O objetivo é limpar o campo sem detonar nenhuma mina, com a ajuda de pistas sobre o número de minas vizinhas a cada parcela.

1.1 O campo e as parcelas

O **campo** do jogo das minas é uma estrutura retangular formada por $N_c \times N_l$ parcelas. Cada parcela é definida por uma **coordenada** no campo, sendo N_c o número de colunas e N_l o número de linhas. As colunas são identificadas por letras maiúsculas de `A` até `Z`, no máximo. As linhas são identificadas por um número inteiro do 1 até 99, no máximo.

As **parcelas** do campo de minas podem esconder ou não uma mina. Adicionalmente, cada parcela pode-se encontrar em um de três estados possíveis: *tapada*, *limpa* ou *marcada* com uma bandeira.

O exemplo da Figura 1 mostra um campo de minas de tamanho 10×8, com parcelas tapadas (verde), parcelas limpas (castanho) e parcelas marcadas (bandeira vermelha). Os números nas parcelas limpas são as pistas que indicam o número de minas que se encontram em parcelas vizinhas (os zeros são omitidos). Consideram-se parcelas vizinhas de uma parcela todas as parcelas que se encontram imediatamente acima, abaixo, à esquerda, à direita, ou em qualquer diagonal dela.

1.2 Regras do jogo

No jogo das minas todas as parcelas do campo de minas se encontram inicialmente tapadas. Em sucessivos turnos, o jogador pode escolher uma de duas ações: limpar ou marcar com uma bandeira uma parcela tapada. Se o jogador escolhe limpar uma parcela que

[1] `https://en.wikipedia.org/wiki/Minesweeper_(video_game)`

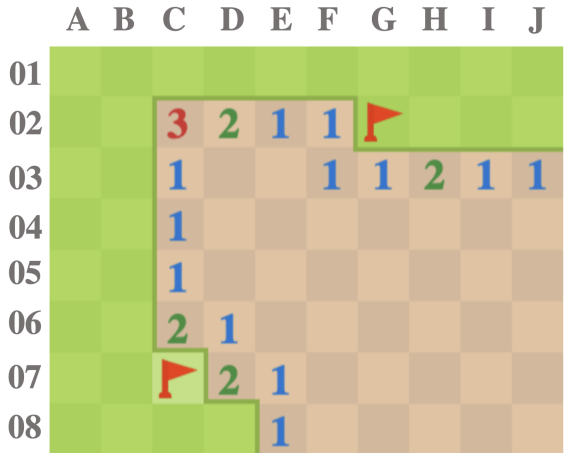

Figura 1: Campo de minas de dimensão 10×8 com pardelas tapadas (verde), parcelas limpas (castanho) e parcelas marcadas (bandeira vermelha). As parcelas limpas mostram as pistas, isto é, o número de parcelas vizinhas com mina.

esconde uma mina, a mina detona e o jogo termina. Caso contrário, a parcela é limpa e mostra uma pista a indicar o número de minas em parcelas vizinhas. Se não existe nenhuma mina em parcelas vizinhas, estas parcelas vizinhas são limpas automaticamente. A limpeza automática de parcelas vizinhas é aplicada de forma iterativa.

A opção de marcar uma parcela tapada com uma bandeira permite ao jogador indicar a (possível) existência de uma mina escondida nessa parcela. As parcelas marcadas são consideradas tapadas, sendo ainda possível retirar a bandeira ou limpar a parcela. No desenvolver do jogo, o jogador deve usar as pistas fornecidas pelas parcelas limpas para deduzir outras parcelas que são seguras para limpar, obtendo iterativamente mais informações para resolver o campo de minas. O jogo também mostra o número de minas totais escondidas nas parcelas do campo de minas e o número total de parcelas marcadas com bandeiras (que pode ser superior ao número de minas). Para ganhar um jogo das minas, o jogador deve limpar todas as parcelas que não escondam minas, sem detonar nenhuma mina no decorrer do jogo.

1.3 Posicionamento inicial das minas

O jogo começa quando o jogador seleciona pela primeira vez uma parcela do campo de minas que deseja limpar. Tal como em algumas versões populares do jogo, é garantido que a primeira parcela limpa não esconde nenhuma mina, assim como nenhuma das suas parcelas vizinhas. Para isso, a geração de quais parcelas contêm minas só deve acontecer depois deste passo. Para gerar estas coordenadas, o jogo utiliza um **gerador** de números pseudoaleatórios. Para cada coordenada, primeiro é obtida a coluna como um caráter pseudoaleatório entre `A` e a última coluna do campo de minas e, a seguir, é obtida a linha da coordenada a partir de um número pseudoaleatório entre 1 e o número de linhas do campo. O processo repete-se até obter tantas coordenadas como pretendidas, que não sejam coincidentes com a parcela inicial, nem com as suas vizinhas, nem com minas previamente colocadas.

1.4 Gerador de números pseudoaleatórios

Os geradores de números pseudoaleatórios tipicamente obtêm sequências de números transformando sucessivamente o seu *estado*, isto é, o número anterior gerado. O primeiro número utilizado para inicializar o estado do gerador é normalmente conhecido como *seed*. Neste projeto, o gerador a implementar será do tipo *xorshift* [2]. Estes geradores têm um ciclo de repetição que depende da dimensão do seu estado e de alguns parâmetros. Para transformar o seu estado, o gerador aplica uma sequência de 3 operações bit a bit sobre o seu estado conhecidas como `xorshift`. A operação `xorshift` consiste num deslocamento de bits ou *shift* (à esquerda ou à direita), seguido de um ou-exclusivo (`xor`) bit a bit do resultado do `shift` com o próprio estado. O primeiro e o último `xorshift` são à esquerda, e o segundo à direita. O número de bits a deslocar depende da dimensão do estado do gerador, 32 ou 64: o gerador de 32 bits utiliza valores $(13, 17, 5)$ para os deslocamentos, enquanto que o de 64 bits $(13, 7, 17)$. Como o Python oferece precisão ilimitada para representar inteiros, de modo a imitar o comportamento do gerador do 32 ou 64 bits, é preciso aplicar um `and` bit-a-bit com o valor `0xFFFFFFFF` (32 bits com valor 1) ou `0xFFFFFFFFFFFFFFFF` (64 bits com valor 1) após cada deslocamento à esquerda. O Algortimo 1 mostra a transformação que é aplicada pelo gerador `xorshift` de 32 bits. Em Python, para as operações de deslocamento são usados os operadores `<<` e `>>`, para o ou-exclusivo bit a bit o símbolo `^`, e para o `and` bit a bit `&`.

```
s ^= (s << 13) & 0xFFFFFFFF
s ^= (s >> 17) & 0xFFFFFFFF
s ^= (s << 5) & 0xFFFFFFFF
```

Algoritmo 1: Sequência de operações `xorshift` do gerador de 32 bits.

2 Trabalho a realizar

Um dos objetivos deste segundo projeto é definir 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 das minas.

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.

[2] `https://en.wikipedia.org/wiki/Xorshift`

- Todas as funções de alto nível (ou seja, que não correspondem a operações básicas) devem respeitar as barreiras de abstração.

2.1.1 TAD *gerador* (1,5 valores)

O TAD *gerador* é usado para representar o estado de um gerador de números pseudoaleatórios *xorshift*. As operações básicas associadas a este TAD são:

- Construtores
  - *cria_gerador*: *int* × *int* ↦ *gerador*
    cria_gerador(b, s) recebe um inteiro b correspondente ao número de bits do gerador e um inteiro positivo s correspondente à *seed* ou estado inicial, e devolve o gerador correspondente. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_gerador: argumentos invalidos'` caso os seus argumentos não sejam válidos.
  - *cria_copia_gerador*: *gerador* ↦ *gerador*
    cria_copia_gerador(g) recebe um *gerador* e devolve uma cópia nova do *gerador*.
- Seletores
  - *obtem_estado*: *gerador* ↦ *int*
    obtem_estado(g) devolve o estado atual do gerador g sem o alterar.
- Modificadores
  - *define_estado*: *gerador* × *int* ↦ *int*
    define_estado(g, s) define o novo valor do estado do gerador g como sendo s, e devolve s.
  - *atualiza_estado*: *gerador* ↦ *int*
    atualiza_estado(g) atualiza o estado do gerador g de acordo com o algoritmo *xorshift* de geração de números pseudoaleatórios, e devolve-o.
- Reconhecedor
  - *eh_gerador*: *universal* ↦ *booleano*
    eh_gerador(arg) devolve `True` caso o seu argumento seja um TAD *gerador* e `False` caso contrário.
- Teste
  - *geradores_iguais*: *gerador* × *gerador* ↦ *booleano*
    geradores_iguais(g1, g2) devolve `True` apenas se $g1$ e $g2$ são geradores e são iguais.

- Transformador
  - *gerador_para_str*: *gerador* $\mapsto$ *str*
    gerador_para_str(g) devolve a cadeia de carateres que representa o seu argumento como mostrado nos exemplos.

As funções de alto nível associadas a este TAD são:

- *gera_numero_aleatorio*: *gerador*$\times$ *int* $\mapsto$ *int*

  gera_numero_aleatorio(g, n) atualiza o estado do gerador g e devolve um número aleatório no intervalo $[1, n]$ obtido a partir do novo estado s de g como `1 + mod(s, n)`, em que `mod()` corresponde à operação resto da divisão inteira.

- *gera_carater_aleatorio*: *gerador*$\times$ *str* $\mapsto$ *str*

  gera_carater_aleatorio(g, c) atualiza o estado do gerador g e devolve um caráter aleatório no intervalo entre `'A'` e o caráter maiúsculo c. Este é obtido a partir do novo estado s de g como o caráter na posição `mod(s, l)` da cadeia de carateres de tamanho l formada por todos os carateres entre `'A'` e c. A operação `mod()` corresponde ao resto da divisão inteira.

Exemplos de interação:

```
>>> g1 = cria_gerador(32, 1)
>>> gerador_para_str(g1)
'xorshift32(s=1)'
>>> [atualiza_estado(g1) for n in range(3)]
[270369, 67634689, 2647435461]
>>> gera_numero_aleatorio(g1, 25)
21
>>> gerador_para_str(g1)
'xorshift32(s=307599695)'
>>> g2 = cria_gerador(64, 1)
>>> [atualiza_estado(g2) for n in range(5)]
[1082269761, 1152992998833853505, 11177516664432764457,
 17678023832001937445, 9659130143999365733]
>>> gerador_para_str(g2)
'xorshift64(s=9659130143999365733)'
>>> gera_carater_aleatorio(g2, 'Z')
'L'
```

2.1.2 TAD *coordenada* (2,0 valores)

O TAD imutável *coordenada* é usado para representar a coordenada que ocupa uma parcela em um campo de minas. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_coordenada*: *str* × *int* ↦ *coordenada*
    cria_coordenada(col, lin) recebe os valores correspondentes à coluna *col* e linha *lin* e devolve a coordenada correspondente. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_coordenada: argumentos invalidos'` caso os seus argumentos não sejam válidos.
- Seletores
  - *obtem_coluna*: *coordenada* ↦ *str*
    obtem_coluna(c) devolve a coluna *col* da coordenada *c*.
  - *obtem_linha*: *coordenada* ↦ *int*
    obtem_linha(c) devolve a linha *lin* da coordenada *c*.
- Reconhecedor
  - *eh_coordenada*: *universal* ↦ *booleano*
    eh_coordenada(arg) devolve `True` caso o seu argumento seja um TAD *coordenada* e `False` caso contrário.
- Teste
  - *coordenadas_iguais*: *coordenada* × *coordenada* ↦ *booleano*
    coordenadas_iguais(c1, c2) devolve `True` apenas se $c1$ e $c2$ são coordenadas e são iguais.
- Transformador
  - *coordenada_para_str*: *coordenada* ↦ *str*
    coordenada_para_str(c) devolve a cadeia de carateres que representa o seu argumento, como mostrado nos exemplos.
  - *str_para_coordenada*: *str* ↦ *coordenada*
    str_para_coordenada(s) devolve a coordenada reapresentada pelo seu argumento.

As funções de alto nível associadas a este TAD são:

- *obtem_coordenadas_vizinhas*: *coordenada* ↦ *tuplo*
  obtem_coordenadas_vizinhas(c) devolve um *tuplo* com as coordenadas vizinhas à coordenada *c*, começando pela coordenada na diagonal acima-esquerda de *c* e seguindo no sentido horário.

- *obtem_coordenada_aleatoria*: *coordenada* × *gerador* ↦ *coordenada*

  obtem_coordenada_aleatoria(c, g) recebe uma coordenada *c* e um TAD *gerador g*, e devolve uma coordenada gerada aleatoriamente como descrito anteriormente em que *c* define a maior coluna e maior linha possíveis. Deve ser gerada, em sequência, primeiro a coluna e depois a linha da coordenada resultado.

Exemplos de interação:

```
>>> c1 = cria_coordenada('A', 200)
Traceback (most recent call last): <...>
ValueError: cria_coordenada: argumentos invalidos
>>> c1 = cria_coordenada('B', 1)
>>> c2 = cria_coordenada('N', 20)
>>> coordenadas_iguais(c1, c2)
False
>>> coordenada_para_str(c1)
'B01'
>>> t = obtem_coordenadas_vizinhas(c1)
>>> tuple(coordenada_para_str(p) for p in t)
('C01', 'C02', 'B02', 'A02', 'A01')
>>> g1 = cria_gerador(32, 1)
>>> c3 = cria_coordenada('Z', 99)
>>> c4 = obtem_coordenada_aleatoria(c3, g1)
>>> coordenada_para_str(c4)
'V68'
```

2.1.3 TAD *parcela* (1,5 valores)

O TAD *parcela* é usado para representar as parcelas de um campo do jogo das minas. As parcelas são caracterizadas pela seu estado (`tapada`, `limpa` ou `marcada`) e podem esconder uma mina. As operações básicas associadas a este tipo são:

- Construtor
  - *cria_parcela*: {} ↦ *parcela*
    cria_parcela() devolve uma parcela tapada sem mina escondida.
  - *cria_copia_parcela*: *parcela* ↦ *parcela*
    cria_copia_parcela(p) recebe uma *parcela p* e devolve uma nova cópia da *parcela*.
- Modificadores
  - *limpa_parcela*: *parcela* ↦ *parcela*
    limpa_parcela(p) modifica destrutivamente a *parcela p* modificando o seu estado para limpa, e devolve a própria *parcela*.

- *marca_parcela*: *parcela* $\mapsto$ *parcela*
  marca_parcela(p) modifica destrutivamente a *parcela p* modificando o seu estado para marcada com uma bandeira, e devolve a própria *parcela*.
- *desmarca_parcela*: *parcela* $\mapsto$ *parcela*
  desmarca_parcela(p) modifica destrutivamente a *parcela p* modificando o seu estado para tapada, e devolve a própria *parcela*.
- *esconde_mina*: *parcela* $\mapsto$ *parcela*
  esconde_mina(p) modifica destrutivamente a *parcela p* escondendo uma mina na parcela, e devolve a própria *parcela*.

- Reconhecedor
  - *eh_parcela*: *universal* $\mapsto$ *booleano*
    eh_parcela(arg) devolve `True` caso o seu argumento seja um TAD *parcela* e `False` caso contrário.
  - *eh_parcela_tapada*: *parcela* $\mapsto$ *booleano*
    eh_parcela_tapada(p) devolve `True` caso a parcela p se encontre tapada e `False` caso contrário.
  - *eh_parcela_marcada*: *parcela* $\mapsto$ *booleano*
    eh_parcela_marcada(p) devolve `True` caso a parcela p se encontre marcada com uma bandeira e `False` caso contrário.
  - *eh_parcela_limpa*: *parcela* $\mapsto$ *booleano*
    eh_parcela_limpa(p) devolve `True` caso a parcela p se encontre limpa e `False` caso contrário.
  - *eh_parcela_minada*: *parcela* $\mapsto$ *booleano*
    eh_parcela_minada(p) devolve `True` caso a parcela p esconda uma mina e `False` caso contrário.

- Teste
  - *parcelas_iguais*: *parcela* $\times$ *parcela* $\mapsto$ *booleano*
    parcelas_iguais(p1, p2) devolve `True` apenas se $p1$ e $p2$ são parcelas e são iguais.

- Transformadores
  - *parcela_para_str*: *parcela* $\mapsto$ *str*
    parcela_para_str(p) devolve a cadeia de caracteres que representa a *parcela* em função do seu estado: parcelas tapadas (`'#'`), parcelas marcadas (`'@'`), parcelas limpas sem mina (`'?'`) e parcelas limpas com mina (`'X'`).

As funções de alto nível associadas a este TAD são:

- *alterna_bandeira*: *parcela* $\mapsto$ *booleano*

  alterna_bandeira(p) recebe uma parcela p e modifica-a destrutivamente da seguinte forma: desmarca se estiver marcada e marca se estiver tapada, devolvendo `True`. Em qualquer outro caso, não modifica a parcela e devolve `False`.

Exemplos de interação:

```
>>> p1 = cria_parcela()
>>> p2 = cria_copia_parcela(p1)
>>> parcela_para_str(p1)
'#'
>>> parcela_para_str(limpa_parcela(p1))
'?'
>>> parcelas_iguais(p1, p2)
False
>>> parcela_para_str(esconde_mina(p2))
'#'
>>> alterna_bandeira(p2)
True
>>> parcela_para_str(p2)
'@'
>>> alterna_bandeira(p1)
False
>>> eh_parcela_minada(p2)
True
```

2.1.4 TAD *campo* (4,0 valores)

O TAD *campo* é usado para representar o campo de minas do jogo das minas. As operações básicas associadas a este TAD são:

- Construtor

  - *cria_campo*: *str* $\times$ *int* $\mapsto$ *campo*

    cria_campo(c, l) recebe uma cadeia de carateres e um inteiro correspondentes à última coluna e à última linha de um campo de minas, e devolve o campo do tamanho pretendido formado por parcelas tapadas sem minas. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_campo: argumentos invalidos'` caso os seus argumentos não sejam válidos.

  - *cria_copia_campo*: *campo* $\mapsto$ *campo*

    cria_copia_campo(m) recebe um *campo* e devolve uma nova cópia do *campo*.

- Seletores
  - *obtem_ultima_coluna*: *campo* $\mapsto$ *str*
    obtem_ultima_coluna(m) devolve a cadeia de caracteres que corresponde à última coluna do campo de minas.
  - *obtem_ultima_linha*: *campo* $\mapsto$ *int*
    obtem_ultima_linha(m) devolve o valor inteiro que corresponde à última linha do campo de minas.
  - *obtem_parcela*: *campo* $\times$ *coordenada* $\mapsto$ *parcela*
    obtem_parcela(m, c) devolve a parcela do campo m que se encontra na coordenada c.
  - *obtem_coordenadas*: *campo*$\times$ *str* $\mapsto$ *tuplo*
    obtem_coordenadas(m, s) devolve o tuplo formado pelas coordenadas ordenadas em ordem ascendente de esquerda à direita e de cima a baixo das parcelas dependendo do valor de s: `'limpas'` para as parcelas limpas, `'tapadas'` para as parcelas tapadas, `'marcadas'` para as parcelas marcadas, e `'minadas'` para as parcelas que escondem minas.
  - *obtem_numero_minas_vizinhas*: *campo*$\times$ *coordenada* $\mapsto$ *int*
    obtem_numero_minas_vizinhas(m, c) devolve o número de parcelas vizinhas da parcela na coordenada c que escondem uma mina.
- Reconhecedores
  - *eh_campo*: *universal* $\mapsto$ *booleano*
    eh_campo(arg) devolve `True` caso o seu argumento seja um TAD *campo* e `False` caso contrário.
  - *eh_coordenada_do_campo*: *campo* $\times$ *coordenada* $\mapsto$ *booleano*
    eh_coordenada_do_campo(m, c) devolve `True` se c é uma coordenada válida dentro do campo m.
- Teste
  - *campos_iguais*: *campo* $\times$ *campo* $\mapsto$ *booleano*
    campos_iguais(m1, m2) devolve `True` apenas se $m1$ e $m2$ forem campos e forem iguais.
- Transformador
  - *campo_para_str*: *campo* $\mapsto$ *str*
    campo_para_str(m) devolve uma cadeia de caracteres que representa o campo de minas como mostrado nos exemplos.

As funções de alto nível associadas a este TAD são:

- coloca_minas: *campo* × *coordenada* × *gerador* × *int* ↦ *campo*

  coloca_minas(m, c, g, n) modifica destrutivamente o campo m escondendo n minas em parcelas dentro do campo. As n coordenadas são geradas em sequência utilizando o gerador g, de modo a que não coincidam com a coordenada c nem com nenhuma parcela vizinha desta, nem se sobreponham com minas colocadas anteriormente.

- limpa_campo: *campo* × *coordenada* ↦ *campo*

  limpa_campo(m, c) modifica destrutivamente o campo limpando a parcela na coordenada c e o devolvendo-a. Se não houver nenhuma mina vizinha escondida, limpa iterativamente todas as parcelas vizinhas tapadas. Caso a parcela se encontre já limpa, a operação não tem efeito.

Exemplos de interação:

```
>>> m = cria_campo('E',5)
>>> obtem_ultima_coluna(m), obtem_ultima_linha(m)
('E', 5)
>>> campo_para_str(m)
'   ABCDE\n  +-----+\n01|#####|\n02|#####|\n03|#####|\n04|#####|\n
05|#####|\n  +-----+'
>>> for l in 'ABC':esconde_mina(obtem_parcela(m, cria_coordenada(l,1)))
>>> for l in 'BC':esconde_mina(obtem_parcela(m, cria_coordenada(l,2)))
>>> for l in 'DE':limpa_parcela(obtem_parcela(m, cria_coordenada(l,1)))
>>> for l in 'AD':limpa_parcela(obtem_parcela(m, cria_coordenada(l,2)))
>>> for l in 'ABCDE':limpa_parcela(
                    obtem_parcela(m, cria_coordenada(l,3)))
>>> alterna_bandeira(obtem_parcela(m, cria_coordenada('D',4)))
True
>>> print(campo_para_str(m))
   ABCDE
  +-----+
01|###2 |
02|3##2#|
03|1221 |
04|###@#|
05|#####|
  +-----+
>>> print(campo_para_str(limpa_campo(m, cria_coordenada('A', 5))))
   ABCDE
  +-----+
01|###2 |
```

```
02|3##2#|
03|1221 |
04|   @ |
05|     |
  +-----+
>>> m = cria_campo('E',5)
>>> g = cria_gerador(32, 1)
>>> c = cria_coordenada('D', 4)
>>> m = coloca_minas(m, c, g, 2)
>>> tuple(coordenada_para_str(p) for p in
                    obtem_coordenadas(m, 'minadas'))
('B01', 'C01')
>>> print(campo_para_str(limpa_campo(m, c)))
   ABCDE
  +-----+
01|###1 |
02|1221 |
03|     |
04|     |
05|     |
  +-----+
```

2.2 Funções adicionais

2.2.1 jogo_ganho: *campo* $\mapsto$ *booleano* (0,5 valores)

jogo_ganho(m) é uma função auxiliar que recebe um campo do jogo das minas e devolve `True` se todas as parcelas sem minas se encontram limpas, ou `False` caso contrário.

```
>>> m = cria_campo('F',6)
>>> jogo_ganho(m)
False
>>> g = cria_gerador(32, 2)
>>> c = cria_coordenada('D', 4)
>>> m = coloca_minas(m, c, g, 1)
>>> print(campo_para_str(limpa_campo(m, c)))
   ABCDEF
  +------+
01|#1    |
02|11    |
03|      |
04|      |
05|      |
```

```
06|      |
  +------+
>>> jogo_ganho(m)
True
```

2.2.2 turno_jogador: *campo* $\mapsto$ *booleano* (1,0 valores)

turno_jogador(m) é uma função auxiliar que recebe um campo de minas e oferece ao jogador a opção de escolher uma ação e uma coordenada. A função modifica destrutivamente o campo de acordo com ação escolhida, devolvendo `False` caso o jogador tenha limpo uma parcela que continha uma mina, ou `True` caso contrário. A função deve apresentar a mensagem `'Escolha uma ação, [L]impar ou [M]arcar:'`, para pedir ao utilizador que escolha uma ação. A função deve repetir a mensagem até o jogador introduzir uma ação válida (`'L'` ou `'M'`). A seguir, a função deve apresentar a mensagem `'Escolha uma coordenada:'` para selecionar a coordenada do campo que se pretende limpar ou marcar (desmarcar). Tal como para a ação, a função deve repetir a mensagem até o jogador introduzir a representação externa de uma coordenada do campo de minas.

```
>>> m = cria_campo('M',5)
>>> g = cria_gerador(32, 2)
>>> c = cria_coordenada('G', 3)
>>> m = coloca_minas(m, c, g, 5)
>>> turno_jogador(m)
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:G03
True
>>> print(campo_para_str(m))
   ABCDEFGHIJKLM
  +-------------+
01|####1  1###1 |
02|11111  113#2 |
03|         2#2 |
04|         111 |
05|             |
  +-------------+
```

2.2.3 minas: *str* $\times$ *int* $\times$ *int* $\times$ *int* $\times$ *int* $\mapsto$ *booleano* (1,5 valores)

minas(c, l, n, d, s) é a função principal que permite jogar ao jogo das minas. A função recebe uma cadeia de carateres e 4 valores inteiros correspondentes, respetivamente, a: última coluna c; última linha l; número de parcelas com minas n; dimensão do gerador de números d; e estado inicial ou *seed* s para a geração de números aleatórios. A função devolve `True` se o jogador conseguir ganhar o jogo, ou `False` caso contrário. A função

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

Exemplo 1

```
>>> minas('Z', 5, 6, 32, 2)
   [Bandeiras 0/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +--------------------------+
01|##########################|
02|##########################|
03|##########################|
04|##########################|
05|##########################|
  +--------------------------+
Escolha uma coordenada:M03
   [Bandeiras 0/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +--------------------------+
01|#1       1#1   1#1  1#####|
02|11       2#2   111  111###|
03|         2#2          1###|
04|         111          1###|
05|                      1###|
  +--------------------------+
Escolha uma ação, [L]impar ou [M]arcar:M
Escolha uma coordenada:V01
   [Bandeiras 1/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +--------------------------+
01|#1       1#1   1#1  1@####|
02|11       2#2   111  111###|
03|         2#2          1###|
04|         111          1###|
05|                      1###|
  +--------------------------+
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:W01
   [Bandeiras 1/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +--------------------------+
01|#1       1#1   1#1  1@1###|
02|11       2#2   111  111###|
03|         2#2          1###|
04|         111          1###|
```

```
05|                      1###|
  +-------------------------+
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:X01
   [Bandeiras 1/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|#1       1#1   1#1  1@1   |
02|11       2#2   111  111   |
03|         2#2          111 |
04|         111          1#1 |
05|                      1#1 |
  +-------------------------+
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:X05
   [Bandeiras 1/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|#1       1#1   1#1  1@1   |
02|11       2#2   111  111   |
03|         2#2          111 |
04|         111          1#1 |
05|                      111 |
  +-------------------------+
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:K01
   [Bandeiras 1/6]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|#1       111   1#1  1@1   |
02|11       2#2   111  111   |
03|         2#2          111 |
04|         111          1#1 |
05|                      111 |
  +-------------------------+
VITORIA!!!
True
```

Exemplo 2

```
>>> minas('Z', 5, 10, 32, 15)
   [Bandeiras 0/10]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
  +-------------------------+
01|#########################|
02|#########################|
03|#########################|
04|#########################|
05|#########################|
  +-------------------------+
Escolha uma coordenada:M03
   [Bandeiras 0/10]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|        1#1     1#########|
02| 111    1#1   112#########|
03| 1#21   111  12###########|
04| 12#1        1###########|
05|  1#1        1###########|
  +-------------------------+
Escolha uma ação, [L]impar ou [M]arcar:M
Escolha uma coordenada:C3
Escolha uma coordenada:C03
   [Bandeiras 1/10]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|        1#1     1#########|
02| 111    1#1   112#########|
03| 1@21   111  12###########|
04| 12#1        1###########|
05|  1#1        1###########|
  +-------------------------+
Escolha uma ação, [L]impar ou [M]arcar:L
Escolha uma coordenada:D04
   [Bandeiras 1/10]
   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  +-------------------------+
01|        1#1     1#########|
02| 111    1#1   112#########|
03| 1@21   111  12###########|
04| 12X1        1###########|
05|  1#1        1###########|
  +-------------------------+
BOOOOOOOM!!!
False
```

3 Condições de Realização e Prazos

- A entrega do 2º projeto será efetuada exclusivamente por via eletrónica. Deverá submeter o seu projeto através do sistema Mooshak, até às **17:00 do dia 11 de Novembro de 2022**. Depois desta hora, não serão aceites projetos sob pretexto algum.
- Deverá submeter um único ficheiro com extensão *.py* contendo todo o código do seu projeto.
- O sistema de submissão assume que o ficheiro está codificado em UTF-8. Alguns editores podem utilizar uma codificação diferente, ou a utilização de alguns carateres mais estranhos (nomeadamente nos comentários) não representáveis em UTF-8 pode levar a outra codificação. Se todos os testes falharem, pode ser um problema da codificação usada. Nesse caso, deverá especificar qual é a codificação do ficheiro na primeira linha deste[3].
- 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[4], o corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

4 Submissão

A submissão e avaliação da execução do projeto de FP é feita utilizando o sistema Mooshak[5]. Para obter as necessárias credenciais de acesso e poder usar o sistema deverá:

- Se já obteve senha para acesso ao sistema no primeiro projeto, deve utilizar a mesma senha. Caso contrário, obtenha uma senha seguindo as instruções na página: `http://acm.tecnico.ulisboa.pt/~fpshak/cgi-bin/getpass-fp22`. A senha ser-lhe-á enviada para o email que tem configurado no Fenix. Se a senha não lhe chegar de imediato, aguarde.

[3] `https://www.python.org/dev/peps/pep-0263/`
[4] `https://theory.stanford.edu/~aiken/moss`
[5] A versão de Python utilizada nos testes automáticos é Python 3.7.3.

- Após ter recebido a sua password por email, deve efetuar o login no sistema através da página: `http://acm.tecnico.ulisboa.pt/~fpshak/`. Preencha os campos com a informação fornecida no email.

- 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 (20-30 seg) para que o sistema processe a sua submissão!!!

- Quando a submissão tiver sido processada, poderá visualizar na tabela o resultado correspondente. Receberá no seu email um relatório de execução com os detalhes da avaliação automática do seu projeto podendo ver o número de testes passados/falhados.

- Para sair do sistema utilize o botão *"Logout"*.

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

- Só poderá efetuar uma nova submissão 5 minutos depois da submissão anterior.

- O sistema só permite 10 submissões em simultâneo pelo que uma submissão poderá ser recusada se este limite for excedido [6].

- Não pode ter submissões duplicadas, ou seja, o sistema pode recusar uma submissão caso seja igual a uma das anteriores.

- 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. Não há excepções!

- Cada aluno tem direito a **15 submissões sem penalização** no Mooshak. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.

5 Classificação

A nota do projeto será baseada nos seguintes aspetos:

[6] Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns alunos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

1. **Execução correta (60%).** A avaliação da correta execução será feita através do sistema Mooshak. O tempo de execução de cada teste está limitado, bem como a memória utilizada.
   Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. O resultado da avaliação automática é calculado em função da fração de testes privados que o programa passa. Como a avaliação automática vale 60% (equivalente a 12 valores) da nota, uma submissão obtém a nota máxima de 1200 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, para 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 com o sistema Mooshak. 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.

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 (*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. Ao desenvolver cada uma das funções pedidas, comece por perceber se pode usar alguma das anteriores.

- 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 ver 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.