TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2021-22
Segundo Projeto
29 de Outubro de 2021

O Prado

O segundo projeto de Fundamentos da Programação consiste em escrever um programa em Python que simule o ecossistema de um prado em que convivem animais que se movimentam, alimentam, reproduzem e morrem. Para este efeito, deverá definir um conjunto de tipos abstratos de dados que deverão ser utilizados para manipular a informação necessária ao decorrer do simulador, bem como um conjunto de funções adicionais.

1 Simulação de ecossistemas

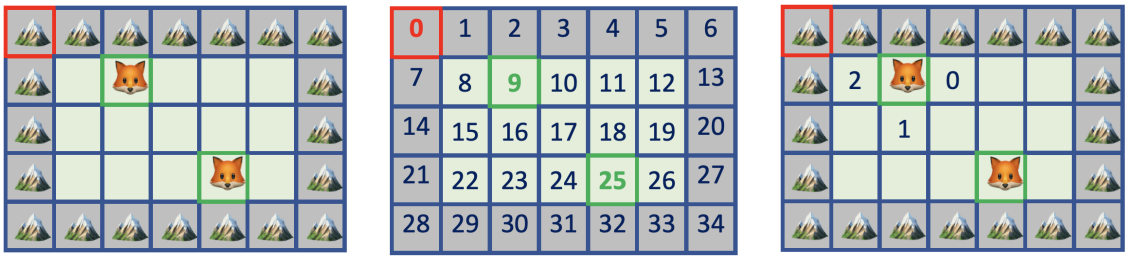
A simulação de ecossistemas decorre num prado rodeado por montanhas. No início, algumas das posições estão ocupadas por animais, que podem ser predadores ou presas, e as restantes estão vazias ou contêm obstáculos. Os animais podem-se movimentar, alimentar, reproduzir e morrer, com regras diferentes para predadores ou presas. A população do prado evolui ao longo de etapas de tempo discretas (gerações) de acordo com estas regras. A simulação consiste na construção de gerações sucessivas da população no prado.

1.1 O prado e os animais

O **prado** é uma estrutura retangular de tamanho $N_x \times N_y$, sendo N_x o tamanho máximo do eixo de abcissas e N_y o tamanho máximo do eixo de ordenadas. Cada posição (x, y) do prado é indexada a partir da posição de origem $(0, 0)$ que corresponde ao canto superior esquerdo do prado. Num prado, todas as posições do limite exterior são *montanhas*, ou seja, correspondem a posições que não podem ser ocupadas. As restantes posições podem corresponder a *espaços vazios*, que são posições passíveis de serem ocupadas por animais que habitam no prado, ou *rochedos*, obstáculos que não podem ser ocupados por nenhum animal. O exemplo da Figura 1a mostra um prado de tamanho 7×5, com montanhas a toda a volta, e com dois animais situados nas posições $(2, 1)$ e $(4, 3)$.

A **ordem de leitura** das posições do prado é sempre feita da esquerda para a direita seguida de cima para baixo, como mostrado na Figura 1b. A cada uma das posições de um prado corresponde um **valor numérico** que respeita a ordem de leitura, ou seja, à posição de origem $(0, 0)$ corresponde o valor 0, e às posições $(2, 1)$ e $(4, 3)$ da Figura 1a, correspondem os valores 9 e 25, respectivamente.

Os **animais** do prado são caracterizados pela idade (gerações passadas desde o seu nascimento ou última reprodução), frequência de reprodução (mínimo número de gerações necessárias para se reproduzir), fome (gerações passadas desde a última alimentação) e frequência de alimentação (máximo número de gerações passíveis sem se alimentar).



a) b) c)

Figura 1: a) Prado de dimensão 7×5 com dois animais situados nas posições $(2, 1)$ e $(4, 3)$. As posições cinzentas correspondem a montanhas ou rochedos enquanto que o resto de posições correspondem a espaços livres do prado. A posição destacada a vermelho indica a origem das coordenadas; b) Ordem de leitura das posições do prado. A vermelho a posição inicial e a verde as posições dos animais; c) Indexação das posições adjacentes livres à posição $(2, 1)$ no sentido dos ponteiros do relógio.

1.2 Regras de evolução

Em cada geração, cada animal de forma sequencial –seguindo a ordem de leitura do prado– realiza o seu turno de ação. No início do turno de cada animal, a sua idade e fome (apenas para os predadores) são incrementadas. A seguir, o animal tenta realizar um movimento e eventualmente pode-se reproduzir, alimentar ou morrer de acordo com as regras seguintes.

1.2.1 Regras de movimento

- Os animais movem-se para cima ou para baixo, para a esquerda ou para a direita, mas não diagonalmente. Os animais avançam no máximo uma posição por geração.
- As montanhas e rochedos não podem ser ocupados por nenhum animal (são muito íngremes para escalar).
- Os **predadores** tentam mover-se para uma posição adjacente que tiver uma presa, comendo-a. Se várias posições adjacentes tiverem presas, então uma é escolhida usando o método descrito abaixo. Se nenhuma posição adjacente tiver presas e se pelo menos uma das posições adjacentes estiver vazia, o predador move-se para lá (escolhendo usando o método descrito abaixo). Caso contrário, permanece no lugar.
- As **presas** tentam mover-se para uma posição vazia adjacente, escolhendo de acordo com o método descrito abaixo se houver várias vazias. Se nenhuma posição adjacente estiver vazia, permanece no lugar.
- As seguintes regras são aplicadas para selecionar uma posição quando múltiplas escolhas são possíveis:

1. Numerar as posições adjacentes possíveis começando de 0, no sentido horário a partir da posição 12:00 (ou seja, para cima, à direita, para baixo, à esquerda) como mostrado na Figura 1c. Apenas as posições que estão desocupadas (para movimentos) ou ocupadas por presas (no caso dos predadores, para os predadores comerem), são numeradas. Chamamos p ao número de possíveis posições.
2. Obter o valor numérico N correspondente à posição atual ocupada pelo animal no prado. Por exemplo, na Figura 1b o valor numérico N do animal na posição $(2, 1)$ é 9.
3. A posição selecionada é aquela com índice igual a $N \pmod p$. Por exemplo, na Figura 1c, $N = 9$, $p = 3$ e $N \pmod p = 9 \pmod 3 = 0$. Neste exemplo, a posição selecionada é a numerada com índice 0, isto é, a posição $(3, 1)$ à direita da posição atual.

1.2.2 Regras de reprodução

Um animal que atingiu a idade de reprodução (considerando a geração atual), quando se move, reproduz-se, deixando para trás na anterior posição um outro animal da mesma espécie, frequência de reprodução e alimentação, com idade igual a 0 e fome igual a 0. A sua própria idade é reinicializada a 0. Os animais não podem procriar se não se moverem nesse turno (e a sua idade não é redefinida até que realmente se reproduzam).

1.2.3 Regras de alimentação e morte

Os predadores comem apenas presas, não outros predadores. Se no fim do turno do animal a fome de um predador é igual à sua frequência de alimentação, o animal morre. Quando um predador se alimenta de uma presa, a sua fome é reinicializada a 0.

As presas alimentam-se de ervas do prado e nunca morrem de fome (isto é, a sua fome e frequência de alimentação é sempre 0), mas morrem quando são comidas por predadores.

2 Trabalho a realizar

O objetivo 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 simulador de ecossistemas.

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

2.1.1 TAD *posicao* (1,5 valores)

O TAD *posicao* é usado para representar uma posição (x, y) de um prado arbitrariamente grande, sendo x e y dois valores inteiros não negativos. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_posicao*: *int* × *int* ↦ *posicao*
    cria_posicao(x,y) recebe os valores correspondentes às coordenadas de uma posição e devolve a posição correspondente. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_posicao: argumentos invalidos'` caso os seus argumentos não sejam válidos.
  - *cria_copia_posicao*: *posicao* ↦ *posicao*
    cria_copia_posicao(p) recebe uma posição e devolve uma cópia nova da posição.
- Seletores
  - *obter_pos_x*: *posicao* ↦ *int*
    obter_pos_x(p) devolve a componente x da posição p.
  - *obter_pos_y*: *posicao* ↦ *int*
    obter_pos_y(p) devolve a componente y da posição p.
- Reconhecedor
  - *eh_posicao*: *universal* ↦ *booleano*
    eh_posicao(arg) devolve `True` caso o seu argumento seja um TAD *posicao* e `False` caso contrário.
- Teste
  - *posicoes_iguais*: *posicao* × *posicao* ↦ *booleano*
    posicoes_iguais(p1, p2) devolve `True` apenas se $p1$ e $p2$ são posições e são iguais.
- Transformador
  - *posicao_para_str*: *posicao* ↦ *str*
    posicao_para_str(p) devolve a cadeia de caracteres `'(x, y)'` que representa o seu argumento, sendo os valores x e y as coordenadas de p.

As funções de alto nível associadas a este TAD são:

- *obter_posicoes_adjacentes*: *posicao* $\mapsto$ *tuplo*

  obter_posicoes_adjacentes(p) devolve um *tuplo* com as posições adjacentes à posição p, começando pela posição acima de p e seguindo no sentido horário.

- *ordenar_posicoes*: *tuplo* $\mapsto$ *tuplo*

  ordenar_posicoes(t) devolve um tuplo contendo as mesmas posições do tuplo fornecido como argumento, ordenadas de acordo com a ordem de leitura do prado.

Exemplos de interação:

```
>>> p1 = cria_posicao(-1, 2)
Traceback (most recent call last): <...>
ValueError: cria_posicao: argumentos invalidos
>>> p1 = cria_posicao(2, 3)
>>> p2 = cria_posicao(7, 0)
>>> posicoes_iguais(p1, p2)
False
>>> posicao_para_str(p1)
'(2, 3)'
>>> t = obter_posicoes_adjacentes(p2)
>>> tuple(posicao_para_str(p) for p in t)
('(8, 0)', '(7, 1)', '(6, 0)')
>>> tuple(posicao_para_str(p) for p in ordenar_posicoes(t))
('(6, 0)', '(8, 0)', '(7, 1)')
```

2.1.2 TAD *animal* (2.5 valores)

O TAD *animal* é usado para representar os animais do simulador de ecossistemas que habitam o prado, existindo de dois tipos: predadores e presas. Os predadores são caracterizados pela espécie, idade, frequência de reprodução, fome e frequência de alimentação. As presas são apenas caracterizadas pela espécie, idade e frequência de reprodução. As operações básicas associadas a este tipo são:

- Construtor

  - *cria_animal*: *str* $\times$ *int* $\times$ *int* $\mapsto$ *animal*

    cria_animal(s, r, a) recebe uma cadeia de caracteres s não vazia correspondente à espécie do animal e dois valores inteiros correspondentes à frequência de reprodução r (maior do que 0) e à frequência de alimentação a (maior ou igual que 0); e devolve o animal. Animais com frequência de alimentação maior que 0 são considerados *predadores*, caso contrário são considerados *presas*. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_animal: argumentos invalidos'` caso os seus argumentos não sejam válidos.

- *cria_copia_animal*: *animal* $\mapsto$ *animal*
  cria_copia_animal(a) recebe um *animal a* (predador ou presa) e devolve uma nova cópia do *animal*.

- Seletores
  - *obter_especie*: *animal* $\mapsto$ *str*
    obter_especie(a) devolve a cadeia de caracteres correspondente à espécie do *animal*.
  - *obter_freq_reproducao*: *animal* $\mapsto$ *int*
    obter_freq_reproducao(a) devolve a frequência de reprodução do *animal a*.
  - *obter_freq_alimentacao*: *animal* $\mapsto$ *int*
    obter_freq_alimentacao(a) devolve a frequência de alimentação do *animal a* (as presas devolvem sempre 0).
  - *obter_idade*: *animal* $\mapsto$ *int*
    obter_idade(a) devolve a idade do *animal a*.
  - *obter_fome*: *animal* $\mapsto$ *int*
    obter_fome(a) devolve a fome do *animal a* (as presas devolvem sempre 0).

- Modificadores
  - *aumenta_idade*: *animal* $\mapsto$ *animal*
    aumenta_idade(a) modifica destrutivamente o *animal a* incrementando o valor da sua idade em uma unidade, e devolve o próprio *animal*.
  - *reset_idade*: *animal* $\mapsto$ *animal*
    reset_idade(a) modifica destrutivamente o *animal a* definindo o valor da sua idade igual a 0, e devolve o próprio *animal*.
  - *aumenta_fome*: *animal* $\mapsto$ *animal*
    aumenta_fome(a) modifica destrutivamente o *animal* predador *a* incrementando o valor da sua fome em uma unidade, e devolve o próprio *animal*. Esta operação não modifica os animais presa.
  - *reset_fome*: *animal* $\mapsto$ *animal*
    reset_fome(a) modifica destrutivamente o *animal* predador *a* definindo o valor da sua fome igual a 0, e devolve o próprio *animal*. Esta operação não modifica os animais presa.

- Reconhecedor
  - *eh_animal*: *universal* $\mapsto$ *booleano*
    eh_animal(arg) devolve `True` caso o seu argumento seja um TAD *animal* e `False` caso contrário.

  - *eh_predador*: *universal* $\mapsto$ *booleano*

    eh_predador(arg) devolve `True` caso o seu argumento seja um TAD *animal* do tipo predador e `False` caso contrário.

  - *eh_presa*: *universal* $\mapsto$ *booleano*

    eh_presa(arg) devolve `True` caso o seu argumento seja um TAD *animal* do tipo presa e `False` caso contrário.

- Teste

  - *animais_iguais*: *animal* $\times$ *animal* $\mapsto$ *booleano*

    animais_iguais(a1, a2) devolve `True` apenas se $a1$ e $a2$ são animais e são iguais.

- Transformadores

  - *animal_para_char*: *animal* $\mapsto$ *str*

    animal_para_char(a) devolve a cadeia de caracteres dum único elemento correspondente ao primeiro carácter da espécie do *animal* passada por argumento, em maiúscula para animais predadores e em minúscula para animais presa.

  - *animal_para_str*: *animal* $\mapsto$ *str*

    animal_para_str(a) devolve a cadeia de caracteres que representa o *animal* como mostrado nos exemplos a seguir.

As funções de alto nível associadas a este TAD são:

- *eh_animal_fertil*: *animal* $\mapsto$ *booleano*

  eh_animal_fertil(a) devolve `True` caso o *animal* a tenha atingido a idade de reprodução e `False` caso contrário.

- *eh_animal_faminto*: *animal* $\mapsto$ *booleano*

  eh_animal_faminto(a) devolve `True` caso o *animal* a tenha atingindo um valor de fome igual ou superior à sua frequência de alimentação e `False` caso contrário. As presas devolvem sempre `False`.

- *reproduz_animal*: *animal* $\mapsto$ *animal*

  reproduz_animal(a) recebe um animal a devolvendo um novo animal da mesma espécie com idade e fome igual a 0, e modificando destrutivamente o animal passado como argumento a alterando a sua idade para 0.

Exemplos de interação:

```
>>> cria_animal('rabbit', -5, 0)
Traceback (most recent call last): <...>
ValueError: cria_animal: argumentos invalidos
>>> r1 = cria_animal('rabbit', 5, 0)
>>> f1 = cria_animal('fox', 20, 10)
>>> animal_para_str(r1)
'rabbit [0/5]'
>>> animal_para_str(f1)
'fox [0/20;0/10]'
>>> animal_para_char(r1)
'r'
>>> animal_para_char(f1)
'F'
>>> f2 = cria_copia_animal(f1)
>>> f2 = aumenta_idade(aumenta_idade(f2))
>>> f2 = aumenta_fome(f2)
>>> animal_para_str(f1)
'fox [0/20;0/10]'
>>> animal_para_str(f2)
'fox [2/20;1/10]'
>>> animais_iguais(f1, f2)
False
>>> f3 = reproduz_animal(f2)
>>> animal_para_str(f2)
'fox [0/20;1/10]'
>>> animal_para_str(f3)
'fox [0/20;0/10]'
```

2.1.3 TAD *prado* (4 valores)

O TAD *prado* é usado para representar o mapa do ecossistema e as animais que se encontram dentro. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_prado*: *posicao* × *tuplo* × *tuplo* × *tuplo* ↦ *prado*
    cria_prado(d, r, a, p) recebe uma posição d correspondente à posição que ocupa a montanha do canto inferior direito do prado, um tuplo r de 0 ou mais posições correspondentes aos rochedos que não são as montanhas dos limites exteriores do prado, um tuplo a de 1 ou mais animais, e um tuplo p da mesma dimensão do tuplo a com as posições correspondentes ocupadas pelos animais; e devolve o prado que representa internamente o mapa e os animais presentes. O construtor verifica a validade dos seus argumentos, gerando um

`ValueError` com a mensagem `'cria_prado: argumentos invalidos'` caso os seus argumentos não sejam válidos.

- *cria_copia_prado*: *prado* $\mapsto$ *prado*
  cria_copia_prado(m) recebe um *prado* e devolve uma nova cópia do *prado*.

- Seletores
  - *obter_tamanho_x*: *prado* $\mapsto$ *int*
    obter_tamanho_x(m) devolve o valor inteiro que corresponde à dimensão N_x do prado.
  - *obter_tamanho_y*: *prado* $\mapsto$ *int*
    obter_tamanho_y(m) devolve o valor inteiro que corresponde à dimensão N_y do prado.
  - *obter_numero_predadores*: *prado* $\mapsto$ *int*
    obter_numero_predadores(m) devolve o número de animais predadores no prado.
  - *obter_numero_presas*: *prado* $\mapsto$ *int*
    obter_numero_presas(m) devolve o número de animais presa no prado.
  - *obter_posicao_animais*: *prado* $\mapsto$ *tuplo posicoes*
    obter_posicao_animais(m) devolve um tuplo contendo as posições do prado ocupadas por animais, ordenadas em ordem de leitura do prado.
  - *obter_animal*: *prado* $\times$ *posicao* $\mapsto$ *animal*
    obter_animal(m, p) devolve o animal do prado que se encontra na posição p.

- Modificadores
  - *eliminar_animal*: *prado* $\times$ *posicao* $\mapsto$ *prado*
    eliminar_animal(m, p) modifica destrutivamente o prado m eliminando o animal da posição p deixando-a livre. Devolve o próprio prado.
  - *mover_animal*: *prado* $\times$ *posicao* $\times$ *posicao* $\mapsto$ *prado*
    mover_animal(m, p1, p2) modifica destrutivamente o prado m movimentando o animal da posição *p1* para a nova posição *p2*, deixando livre a posição onde se encontrava. Devolve o próprio prado.
  - *inserir_animal*: *prado* $\times$ *animal* $\times$ *posicao* $\mapsto$ *prado*
    inserir_animal(m, a, p) modifica destrutivamente o prado m acrescentando na posição p do prado o animal a passado com argumento. Devolve o próprio prado.

- Reconhecedores
  - *eh_prado*: *universal* $\mapsto$ *booleano*
    eh_prado(arg) devolve `True` caso o seu argumento seja um TAD *prado* e `False` caso contrário.

  - *eh_posicao_animal*: *prado* × *posicao* ↦ *booleano*
    eh_posicao_animal(m, p) devolve `True` apenas no caso da posição p do prado estar ocupada por um animal.
  - *eh_posicao_obstaculo*: *prado* × *posicao* ↦ *booleano*
    eh_posicao_obstaculo(m, p) devolve `True` apenas no caso da posição p do prado corresponder a uma montanha ou rochedo.
  - *eh_posicao_livre*: *prado* × *posicao* ↦ *booleano*
    eh_posicao_livre(m, p) devolve `True` apenas no caso da posição p do prado corresponder a um espaço livre (sem animais, nem obstáculos).

- Teste
  - *prados_iguais*: *prado* × *prado* ↦ *booleano*
    prados_iguais(p1, p2) devolve `True` apenas se $p1$ e $p2$ forem prados e forem iguais.

- Transformador
  - *prado_para_str*: *prado* ↦ *str*
    prado_para_str(m) devolve uma cadeia de caracteres que representa o prado como mostrado nos exemplos.

As funções de alto nível associadas a este TAD são:

- obter_valor_numerico: *prado* × *posicao* ↦ *int*
  obter_valor_numerico(m, p) devolve o valor numérico da posição p correspondente à ordem de leitura no prado m.

- obter_movimento: *prado* × *posicao* ↦ *posicao*
  obter_movimento(m, p) devolve a posição seguinte do animal na posição p dentro do prado m de acordo com as regras de movimento dos animais no prado.

Exemplos de interação:

```
>>> dim = cria_posicao(11, 4)
>>> obs = (cria_posicao(4,2), cria_posicao(5,2))
>>> an1 = tuple(cria_animal('rabbit', 5, 0) for i in range(3))
>>> an2 = (cria_animal('lynx', 20, 15),)
>>> pos = tuple(cria_posicao(p[0],p[1]) \
                        for p in ((5,1),(7,2),(10,1),(6,1)))
>>> prado = cria_prado(dim, obs, an1+an2, pos)
>>> obter_tamanho_x(prado), obter_tamanho_y(prado)
(12, 5)
>>> print(prado_para_str(prado))
```

```
+----------+
|....rL...r|
|...@@.r...|
|..........|
+----------+
>>> p1 = cria_posicao(7,2)
>>> p2 = cria_posicao(9,3)
>>> prado = mover_animal(prado, p1, p2)
>>> print(prado_para_str(prado))
+----------+
|....rL...r|
|...@@.....|
|........r.|
+----------+
>>> obter_valor_numerico(prado, cria_posicao(9,3))
45
>>> posicao_para_str(obter_movimento(prado, cria_posicao(5,1)))
'(4, 1)'
>>> posicao_para_str(obter_movimento(prado, cria_posicao(6,1)))
'(5, 1)'
>>> posicao_para_str(obter_movimento(prado, cria_posicao(10,1)))
'(10, 2)'
```

2.2 Funções adicionais

2.2.1 geracao: *prado* $\mapsto$ *prado* (2 valores)

geracao(m) é a função auxiliar que modifica o prado m fornecido como argumento de acordo com a evolução correspondente a uma geração completa, e devolve o próprio prado. Isto é, seguindo a ordem de leitura do prado, cada animal (vivo) realiza o seu turno de ação de acordo com as regras descritas.

```
>>> dim = cria_posicao(11, 4)
>>> obs = (cria_posicao(4,2), cria_posicao(5,2))
>>> an1 = tuple(cria_animal('sheep', 2, 0) for i in range(3))
>>> an2 = (cria_animal('wolf', 10, 3),)
>>> pos = tuple(cria_posicao(p[0],p[1]) \
                        for p in ((2,2),(4,3),(10,2),(3,2)))
>>> prado = cria_prado(dim, obs, an1+an2, pos)
>>> print(prado_para_str(prado))
+----------+
|..........|
|.sW@@....s|
```

```
|...s......|
+----------+
>>> print(prado_para_str(geracao(prado)))
+----------+
|..W.......|
|s..@@.....|
|....s....s|
+----------+
>>> print(prado_para_str(geracao(prado)))
+----------+
|...W......|
|ss.@@....s|
|...ss....s|
+----------+
>>> print(prado_para_str(geracao(prado)))
+----------+
|.........s|
|...@@....s|
|ssss......|
+----------+
```

2.2.2 simula_ecossistema: *str* × *int* × *booleano* ↦ *tuplo* (2 valores)

simula_ecossitema(f, g, v) é a função principal que permite simular o ecossistema de um prado. A função recebe uma cadeia de caracteres f, um valor inteiro g e um valor booleano v e devolve o tuplo de dois elementos correspondentes ao número de predadores e presas no prado no fim da simulação. A cadeia de caracteres f passada por argumento corresponde ao nome do ficheiro de configuração da simulação. O valor inteiro g corresponde ao número de gerações a simular. O argumento booleano v ativa o modo *verboso* (`True`) ou o modo *quiet* (`False`). No modo *quiet* mostra-se pela saída *standard* o prado, o número de animais e o número de geração no início da simulação e após a última geração. No modo *verboso*, após cada geração, mostra-se também o prado, o número de animais e o número de geração, apenas se o número de animais predadores ou presas se tiver alterado. O exemplo seguinte mostra o conteúdo de um ficheiro de configuração `'config.txt'`:

```
(11, 4)
((4, 2), (5, 2))
('mouse', 2, 0, (2, 2))
('mouse', 2, 0, (4, 3))
('mouse', 2, 0, (10, 2))
('lynx', 10, 3, (3, 2))
```

A primeira linha contém a representação externa do canto inferior direito do prado; a segunda a representação externa das posições dos rochedos do prado; o resto das linhas contém um animal diferente do prado caracterizado pelo seu nome, frequência de reprodução e frequência de alimentação, seguido da representação externa da posição que ocupa inicialmente no prado. Pode assumir que o ficheiro de configuração está sempre bem formado.

Exemplo

```
>>> simula_ecossistema('config.txt', 200, False)
Predadores: 1 vs Presas: 3 (Gen. 0)
+----------+
|.........|
|.mL@@....m|
|...m......|
+----------+
Predadores: 0 vs Presas: 28 (Gen. 200)
+----------+
|mmmmmmmmmm|
|mmm@@mmmmm|
|mmmmmmmmmm|
+----------+
(0, 28)
>>> simula_ecossistema('config.txt', 200, True)
Predadores: 1 vs Presas: 3 (Gen. 0)
+----------+
|.........|
|.mL@@....m|
|...m......|
+----------+
Predadores: 1 vs Presas: 6 (Gen. 2)
+----------+
|...L......|
|mm.@@....m|
|...mm....m|
+----------+
Predadores: 0 vs Presas: 6 (Gen. 3)
+----------+
|.........m|
|...@@....m|
|mmmm......|
+----------+
Predadores: 0 vs Presas: 12 (Gen. 4)
+----------+
```

```
|........mm|
|mmm@@....m|
|mmmmm....m|
+----------+
Predadores: 0 vs Presas: 18 (Gen. 6)
+----------+
|mmm....mmm|
|mmm@@..mmm|
|mmmmm....m|
+----------+
Predadores: 0 vs Presas: 20 (Gen. 7)
+----------+
|mmmm..mmmm|
|mmm@@..mmm|
|mmmm.m.m..|
+----------+
Predadores: 0 vs Presas: 28 (Gen. 8)
+----------+
|mmmmmmmmmm|
|mmm@@mmmmm|
|mmmmmmmmmm|
+----------+
(0, 28)
```

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 19 de Novembro de 2021**. 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[1].

[1] `https://www.python.org/dev/peps/pep-0263/`

- 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[2], 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[3]. 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-fp21`. A senha ser-lhe-á enviada para o email que tem configurado no Fenix. Se a senha não lhe chegar de imediato, aguarde.

- 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"*.

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

[3] A versão de Python utilizada nos testes automáticos é Python 3.7.3.

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 [4].
- 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:

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.
   Existem 237 casos de teste configurados no sistema: 28 testes públicos (disponibilizados na página da disciplina) valendo 0 pontos cada e 209 testes privados (não disponibilizados). 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%).** Esta componente da avaliação é feita automaticamente, recorrendo a um conjunto de *scripts* que testam o respeito pelas barreiras de abstração do código desenvolvido pelos alunos.
3. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura[5]. Em particular, serão consideradas as seguintes componentes:

[4]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.

[5]Podem encontrar algumas boas práticas relacionadas em `https://gist.github.com/ruimaranhao/4e18cbe3dad6f68040c32ed6709090a3`

- 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.