TÉCNICO LISBOA

Fundamentos da Programação

Ano lectivo 2013-14, 2º Semestre

Projecto – Segunda Parte[1]

"Liberta o quadrado"

O jogo "*Liberta o quadrado*" é um quebra-cabeças em que o jogador é confrontado com um tabuleiro de 5×4 contendo peças de diversos formatos e cujo objectivo é manipular as mesmas por forma a permitir a remoção da peça maior: um quadrado grande, de cor normalmente vermelha (ver Figura 1).

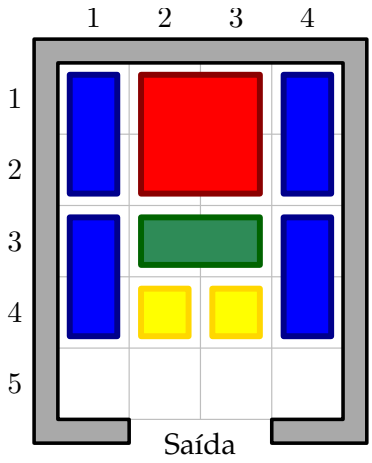


Figura 1: Exemplo de um tabuleiro de "*Liberta o quadrado*".

Para tal, o jogador pode "arrastar" as demais peças para os espaços livres disponíveis, procurando fazer o quadrado vermelho chegar ao fundo do tabuleiro com o mínimo número de movimentos possível.

1 Trabalho a Realizar

O objectivo da segunda parte do projecto é escrever um programa em Python que permita a um utilizador jogar o jogo "*Liberta o quadrado*" no computador. Para tal, deverá definir um conjunto de tipos de informação que deverão ser utilizados para manipular a informação necessária ao decorrer do jogo, bem como um conjunto de funções adicionais que permitirão jogar o jogo propriamente dito.

[1]Tempo estimado de resolução por um grupo de 3 pessoas, após estudo da matéria correspondente: 15 a 20 horas.

1.1 Tipos Abstractos de Informação (TAI)

TAI *coordenada* (1.5 val.)

O TAI *coordenada* será utilizado para indexar as várias posições do tabuleiro. Cada posição do tabuleiro é indexada através da linha respectiva (um inteiro entre 1 e 5) e da coluna respectiva (um inteiro entre 1 e 4). O TAI *coordenada* deverá pois ser um tipo **imutável** que armazena dois inteiros correspondentes a uma linha e uma coluna do tabuleiro.

As operações básicas associadas a este TAI são:

- *cria_coordenada* : *int* $\times$ *int* $\rightarrow$ *coordenada*

  Esta função corresponde ao construtor do tipo *coordenada*. Recebe dois argumentos do tipo inteiro, o primeiro dos quais corresponde a uma linha l (um inteiro entre 1 e 5) e o segundo a uma coluna c (um inteiro entre 1 e 4).

  Deve devolver um elemento do tipo *coordenada* correspondente à posição (l, c).

- *coordenada_linha* : *coordenada* $\rightarrow$ *inteiro*

  Este selector recebe como argumento um elemento do tipo *coordenada* e devolve a linha respectiva.

- *coordenada_coluna* : *coordenada* $\rightarrow$ *inteiro*

  Este selector recebe como argumento um elemento do tipo *coordenada* e devolve a coluna respectiva.

- *e_coordenada* : *universal* $\rightarrow$ *lógico*

  Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *coordenada*, e `False` em caso contrário.

- *coordenadas_iguais* : *coordenada* $\times$ *coordenada* $\rightarrow$ *lógico*

  Este teste recebe como argumentos dois elementos do tipo *coordenada* e devolve `True` caso esses argumentos correspondam à mesma posição (l, c) do tabuleiro, e `False` em caso contrário.

Deve ainda implementar a seguinte função adicional:

- *coordenada_na_direcao* : *coordenada* $\times$ *int* $\times$ *int* $\rightarrow$ *coordenada*

  Esta função recebe como argumentos uma coordenada c e dois inteiros, dx e dy, e devolve as coordenadas da posição que se obtém após um deslocamento de dx linhas e dy colunas a partir da posição correspondente à coordenada c. Deverá considerar os casos especiais seguintes:

  – Caso a posição final fique fora do tabuleiro, a função deverá devolver as coordenadas da posição mais próxima que fique ainda dentro do tabuleiro.

  - Caso a posição final coincida com a posição inicial, a função deverá devolver `False`.

Exemplo de interacção:

```
>>> C1 = cria_coordenada(-1, 1)
[...]
builtins.ValueError: cria_coordenada: argumentos invalidos
>>> C1 = cria_coordenada(1, 5)
[...]
builtins.ValueError: cria_coordenada: argumentos invalidos
>>> C1 = cria_coordenada(0, 0)
[...]
builtins.ValueError: cria_coordenada: argumentos invalidos
>>> C1 = cria_coordenada(1, 2)
>>> coordenada_linha(C1)
1
>>> coordenada_coluna(C1)
2
>>> e_coordenada(0)
False
>>> e_coordenada(C1)
True
>>> C2 = cria_coordenada(2, 1)
>>> coordenadas_iguais(C1, C2)
False
>>> C3 = coordenada_na_direcao(C1, 1, 2)
>>> coordenada_linha(C3)
2
>>> coordenada_coluna(C3)
4
>>> coordenadas_iguais(C1, coordenada_na_direcao(C3, -1, -2))
True
```

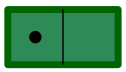
TAI *peca* (3.5 val.)

O TAI *peca* será utilizado para representar as várias formas que preenchem o tabuleiro. Uma peça pode ser de quatro tipos diferentes, representados na Figura 2. Cada peça ocupa uma ou mais posições do tabuleiro, e é caracterizada por um *bloco de referência*, também marcado na Figura 2.

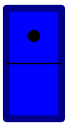
O TAI *peca* deverá ser um tipo **mutável** contendo :

- Informação referente ao tipo de peça, que consiste numa cadeia de caracteres contendo um único caracter, `'U'`,`'S'`,`'I'` ou `'O'`.[2]

[2]Para tornar o programa o mais independente possível da representação escolhida para os tipos de peças, estes devem corresponder a constantes definidas no início do seu programa (`PECA_U`, `PECA_S`, `PECA_I` e `PECA_O`). No ficheiro `proj2aux.py` fornecido pelo corpo docente, estas constantes vêm já definidas.


Forma do tipo U

Forma do tipo S


Forma do tipo I

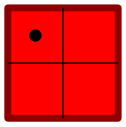

Forma do tipo O

Figura 2: Peças do jogo "*liberta o quadrado*" e respectivos blocos de referência (marcados com "•").

- Uma lista, contendo as coordenadas ocupadas pela peça.

As operações básicas associadas a este TAI são:

- *cria_peca* : *cad. caracteres* × *coordenada* → *peca*

  Esta função corresponde ao construtor do tipo *peca*. Recebe como argumento uma cadeia de caracteres correspondente ao tipo de peça e um elemento do tipo *coordenada*, contendo a coordenada do bloco de referência da peça. A cadeia de caracteres deverá conter um único carácter, correspondente ao tipo de peça em questão (`'I'` para uma peça do tipo I, `'U'` para uma peça do tipo U, `'O'` para uma peça do tipo O e `'S'` para uma peça do tipo S).[3] Caso a cadeia de caracteres não esteja de acordo com a especificação acima, a sua função deverá gerar um erro de valor (`ValueError`) com a mensagem "`cria_peca: argumentos invalidos`".

  A coordenada fornecida deverá ser tal que a peça fique completamente dentro do tabuleiro. Caso contrário, a sua função deverá gerar um erro de valor (`ValueError`) com a mensagem "`cria_peca: coordenada invalida`".

- *peca_posicoes* : *peca* → *lista*

  Este selector recebe como argumento um elemento p do tipo *peca* e devolve uma lista contendo as coordenadas de todas as posições ocupadas pela peça p.

- *peca_tipo* : *peca* → *cad. caracteres*

  Este selector recebe como argumento um elemento p do tipo *peca* e devolve a cadeia de caracteres correspondente tipo da peça p.

- *e_peca* : *universal* → *logico*

[3] De forma equivalente, a cadeia de caracteres deverá corresponder a uma das quatro constantes, `PECA_U`, `PECA_S`, `PECA_I` ou `PECA_O`.

Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *peca*, e `False` em caso contrário.

- *peca_na_posicao* : *peca* $\times$ *coordenada* $\rightarrow$ *logico*

  Este reconhecedor recebe como argumentos um elemento p do tipo *peca* e um elemento c do tipo *coordenada* e devolve `True` caso a peça p ocupe a posição correspondente à coordenada c, e `False` em caso contrário.

- *peca_move* : *peca* $\times$ *int* $\times$ *int* $\rightarrow \{\}$

  Este modificador recebe como argumentos um elemento p do tipo *peca* e dois inteiros, dx e dy, e modifica a posição da peça p, deslocando-a dx linhas e dy colunas. Caso a deslocação (dx, dy) implique que p termina fora do tabuleiro, a sua função deverá gerar um erro de valor (`ValueError`) com a mensagem "`peca_move: argumentos invalidos`".

Deve ainda implementar a seguinte função adicional:

- *posicoes_adjacentes* : *peca* $\times$ *cad. caracteres* $\rightarrow$ *lista*

  Esta função recebe como argumentos um elemento p do tipo *peca* e uma cadeia de caracteres d, correspondendo a uma das quatro direcções (cada direcção é representada por um dos caracteres `'N'`,`'S'`,`'E'` ou `'W'`), e devolve uma lista contendo as coordenadas de todas as posições do tabuleiro adjacentes à peça p na direcção d.[4]

Exemplo de interacção:

```
>>> P = cria_peca('A', cria_coordenada(1, 1))
[...]
builtins.ValueError: cria_peca: argumentos invalidos
>>> P = cria_peca(PECA_O, cria_coordenada(1, 4))
[...]
builtins.ValueError: cria_peca: coordenada invalida
>>> P = cria_peca(PECA_I, cria_coordenada(1,1))
>>> peca_tipo(P)
'I'
>>> L = peca_posicoes(P)
>>> for c in L:
...     print('Linha:', coordenada_linha(c), \
            'Coluna:', coordenada_coluna(c))
...
Linha: 1 Coluna: 1
Linha: 2 Coluna: 1
>>> e_peca(0)
False
>>> e_peca(P)
```

[4]Tal como acontece com a representação do tipo de peças, para tornar o programa o mais independente possível da representação escolhida para as direcções, estas devem ser definidas como constantes no início do seu programa, como indicado no ficheiro `proj2aux.py` fornecido pelo corpo docente.

```
True
>>> peca_na_posicao(P, cria_coordenada(1,2))
False
>>> peca_na_posicao(P, cria_coordenada(2,1))
True
>>> peca_move(P, -1, -1)
[...]
builtins.ValueError: peca_move: argumentos invalidos
>>> peca_move(P, 1, 1)
>>> L = peca_posicoes(P)


>>> for c in L:
...     print('Linha:', coordenada_linha(c), \
...           'Coluna:', coordenada_coluna(c))
...
Linha: 2 Coluna: 2
Linha: 3 Coluna: 2
>>> L = posicoes_adjacentes(P, ESTE)
>>> for c in L:
...     print('Linha:', coordenada_linha(c), \
...           'Coluna:', coordenada_coluna(c))
...
Linha: 2 Coluna: 3
Linha: 3 Coluna: 3
```

TAI *tabuleiro* (3 val.)

O TAI *tabuleiro* será utilizado para armazenar a informação relativa às peças no tabuleiro. As operações básicas associadas a este TAI são:

- *cria_tabuleiro* : *cad. caracteres* $\rightarrow$ *tabuleiro*

  Esta função corresponde ao construtor do tipo *tabuleiro*. Recebe como argumento uma cadeia de caracteres contendo o nome de um ficheiro de texto contendo a disposição inicial das peças no tabuleiro. O construtor deve, pois, abrir, ler e processar o ficheiro respectivo, devolvendo depois um elemento do tipo tabuleiro de acordo com a representação interna escolhida.

  Os ficheiros usados contêm cinco linhas, cada uma com 4 caracteres, correspondendo às várias posições do tabuleiro. Cada carácter corresponde ao tipo de peça na posição respectiva (`'I'` para uma peça do tipo I, `'U'` para uma peça do tipo U, `'O'` para uma peça do tipo O e `'S'` para uma peça do tipo S) ou a um espaço (representado pelo carácter `'X'`). Por exemplo, o tabuleiro da Figura 1 poderia ser representado por um ficheiro com o seguinte conteúdo:

  ```
  IOOI
  IOOI
  IUUI
  ISSI
  ```

```
XXXX
```

No seu programa pode assumir que os ficheiros usados estão sempre de acordo com o formato especificado.

- *tabuleiro_posicao* : *tabuleiro* × *coordenada* → *peca*

  Este selector recebe como argumentos um elemento t do tipo *tabuleiro* e um elemento c do tipo coordenada e devolve um elemento do tipo *peca*, que corresponde à peça que ocupa a posição correspondente à coordenada c. Caso a posição correspondente a c esteja vazia, deverá devolver a cadeia de caracteres que representa o espaço (isto é, o carácter `'X'`).

- *actualiza_tabuleiro* : *tabuleiro* × *coordenada* × *cad. caracteres* → {}

  Este modificador recebe como argumentos um elemento t do tipo *tabuleiro*, um elemento c do tipo *coordenada* e uma cadeia de caracteres d correspondente a uma das 4 direcções e actualiza a informação das peças no tabuleiro t após mover a peça na posição correspondente à coordenada c uma posição na direcção d.

- *tabuleiro_posicao_livre* : *tabuleiro* × *coordenada* → *logica*

  Este reconhecedor recebe como argumentos um elemento t do tipo *tabuleiro* e um elemento c do tipo *coordenada* e devolve `True` caso a posição correspondente à coordenada c esteja livre no tabuleiro t, e `False` em caso contrário.

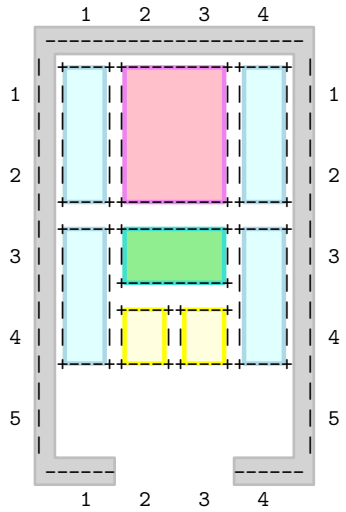
- *e_tabuleiro : universal* → *lógico*

  Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *tabuleiro*, e `False` em caso contrário.

Além das operações básicas acima, o corpo docente fornece ainda no ficheiro `proj2aux.py` o código correspondente ao transformador de saída para o TAI *tabuleiro*:

- *desenha_tabuleiro* : *tabuleiro* → {}

  Esta função desenha para o ecrã uma representação de um tabuleiro. Por exemplo, o tabuleiro representado na Figura 1 seria representado como:



Como exemplo de interacção, suponha que o ficheiro `tab1.txt` contém a descrição do tabuleiro apresentado na Figura 1. De acordo com a especificação já descrita, este ficheiro teria o conteúdo:

```
IOOI
IOOI
IUUI
ISSI
XXXX
```

a partir do qual seria possível obter a interacção apresentada de seguida.

```
>>> T = cria_tabuleiro(0)
[...]
builtins.ValueError: cria_tabuleiro: argumentos invalidos
>>> T = cria_tabuleiro('tab1.txt')
>>> desenha_tabuleiro(T)

      1    2    3    4
   ----------------------
  | +---++--------++---+ |
1 | |   ||        ||   | | 1
  | |   ||        ||   | |
  | |   ||        ||   | |
2 | |   ||        ||   | | 2
  | +---++--------++---+ |
  | +---++--------++---+ |
3 | |   ||        ||   | | 3
  | |   |+--------+|   | |
  | |   |+---++---+|   | |
4 | |   ||   ||   ||   | | 4
  | +---++---++---++---+ |
  |                       |
5 |                       | 5
  |                       |
   ------          ------
      1    2    3    4
>>> P = tabuleiro_posicao(T, cria_coordenada(4, 4))
>>> L = peca_posicoes(P)
>>> for c in L:
...     print('Linha:', coordenada_linha(c), \
...           'Coluna:', coordenada_coluna(c))
...
Linha: 3 Coluna: 4
Linha: 4 Coluna: 4
>>> actualiza_tabuleiro(T, 0, SUL)
[...]
builtins.ValueError: actualiza_tabuleiro: argumentos invalidos
>>> actualiza_tabuleiro(T, cria_coordenada(4, 4), SUL)
>>> desenha_tabuleiro(T)
```

```
      1    2    3    4
    ----------------------
   | +---++--------++---+ |
1  | |   ||        ||   | | 1
   | |   ||        ||   | |
   | |   ||        ||   | |
2  | |   ||        ||   | | 2
   | +---++--------++---+ |
   | +---++--------+      |
3  | |   ||        |      | 3
   | |   |+--------+      |
   | |   |+---++---++---+ |
4  | |   ||   ||   ||   | | 4
   | +---++---++---+|   | |
   |                |   | |
5  |                |   | | 5
   |                +---+ |
    ------          ------
      1    2    3    4
>>> tabuleiro_posicao_livre(T, cria_coordenada(3, 4))
True
>>> tabuleiro_posicao_livre(T, cria_coordenada(5, 4))
False
>>> e_tabuleiro(T)
True
>>> e_tabuleiro(0)
False
```

1.2 Funções Adicionais

- *jogada_valida* : *tabuleiro* × *coordenada* × *cad. caracteres* → *lógico* **(2 val.)**

  Esta função recebe como argumentos um elemento t do tipo *tabuleiro*, um elemento c do tipo *coordenada* e uma cadeia de caracteres d correspondente a uma das 4 direcções. A função devolve `True` se for possível mover a peça que ocupa a posição correspondente à coordenada c na direcção indicada por d, e `False` em caso contrário. Caso não exista nenhuma peça na posição indicada por c, a função deve devolver `False`.

  Exemplo (usando o ficheiro `tab1.txt` do exemplo acima):

```
>>> T = cria_tabuleiro('tab1.txt')
>>> jogada_valida(T, cria_coordenada(1, 1), 'A')
[...]
builtins.ValueError: jogada_valida: argumentos invalidos
>>> jogada_valida(T, cria_coordenada(1, 1), NORTE)
False
>>> jogada_valida(T, cria_coordenada(1, 1), SUL)
```

```
False
>>> jogada_valida(T, cria_coordenada(4, 4), SUL)
True
>>> jogada_valida(T, cria_coordenada(5, 4), SUL)
False
```

- *pede_jogada* : $\{\} \rightarrow$ *lógico* **(1 val.)**

  Esta função recebe não recebe qualquer argumento, limitando-se a pedir ao utilizador para introduzir (i) uma linha entre 1 e 5; (ii) uma coluna entre 1 e 4; e (iii) uma direcção (N, S, E ou W). Caso algum dos valores introduzidos pelo utilizador seja inválido, a função pedir novamente a informação de jogada ao utilizador. A função devolve um elemento c do tipo *coordenada* com a linha e coluna introduzidas pelo utilizador, e uma cadeia de caracteres correspondente à direcção escolhida pelo utilizador.

  Exemplo:

```
>>> C, D = pede_jogada()
Introduza uma linha (1 a 5): 0
Linha invalida.
Introduza uma linha (1 a 5): 1
Introduza uma coluna (1 a 4): 0
Coluna invalida.
Introduza uma linha (1 a 5): 6
Linha invalida.
Introduza uma linha (1 a 5): 1
Introduza uma coluna (1 a 4): 5
Coluna invalida.
Introduza uma linha (1 a 5): 1
Introduza uma coluna (1 a 4): 2
Introduza uma direccao (N, S, E ou W): A
Direcao invalida.
Introduza uma linha (1 a 5): 1
Introduza uma coluna (1 a 4): 4
Introduza uma direccao (N, S, E ou W): S
>>> e_coordenada(C)
True
>>> D
'S'
>>> print('Linha:', coordenada_linha(C), \
...       'Coluna:', coordenada_coluna(C))
Linha: 1 Coluna: 4
```

- *jogo_terminado* : *tabuleiro* $\rightarrow$ *lógico* **(1 val.)**

  Esta função recebe como argumento um elemento t do tipo *tabuleiro* e devolve `True` caso t corresponda a um jogo terminado—isto é, a uma configuração em que a peça do tipo O é adjacente à saída. Devolve `False` em caso contrário.

  Por exemplo, supondo que `tab1.txt` corresponde ao ficheiro

```
IOOI
IOOI
IUUI
ISSI
XXXX
```

e que `tab2.txt` corresponde ao ficheiro

```
ISSI
ISSI
IXXI
IOOI
XOOX
```

teríamos a seguinte interacção:

```
>>> T1 = cria_tabuleiro('tab1.txt')
>>> T2 = cria_tabuleiro('tab2.txt')
>>> jogo_terminado(0)
[...]
builtins.ValueError: jogo_terminado: argumentos invalidos
>>> jogo_terminado(T1)
False
>>> jogo_terminado(T2)
True
```

- *jogo_quadrado* : *cad. caracteres* $\rightarrow \{\}$ **(2 val.)**

  Esta função recebe como argumentos uma cadeia de caracteres, correspondente ao nome de um ficheiro contendo a descrição de um tabuleiro, e permite a um utilizador jogar um jogo completo de "*Liberta o quadrado*".

  Em cada turno, a função deve desenhar o tabuleiro para o ecrã e pedir ao utilizador para introduzir uma nova jogada. Caso a jogada seja válida, deverá actualizar o tabuleiro e repetir este processo até o jogo terminar. Caso contrário, deverá escrever para o ecrã a indicação "`Jogada invalida.`" e solicitar nova jogada ao utilizador.

  Ao terminar o jogo, a função deverá escrever para o ecrã a indicação "`Parabens, terminou o jogo em` k `jogadas.`", em que k corresponde ao numero total de movimentos efectuados pelo jogador.

  Um exemplo de um jogo completo usando o ficheiro `tab1.txt`:

```
IOOI
IOOI
ISSI
ISSI
XXXX
```

  está disponível no site, juntamente com o demais material de apoio ao projecto.

Seguindo a metodologia dos tipos abstractos de informação, *os construtores e os modificadores dos diversos T.A.I.s devem verificar a validade dos seus argumentos*. As restantes

operações básicas (selectores, reconhecedores e testes) não necessitam de efectuar essa verificação.

Todas as demais funções que não sejam operações básicas dos T.A.I.s devem validar os argumentos pela ordem porque aparecem. Em caso de alguma função ser chamada com um (ou mais) argumento(s) inválido(s), a função deve gerar um erro de valor (`ValueError`) com uma mensagem de erro contendo o nome da mesma e a indicação "`argumentos invalidos`" (ver exemplos fornecidos). De notar ainda que algumas das funções efectuam validações adicionais, que foram já incluídas na descrição da própria função, tendo essas validações mensagens de erro específicas.

1.3 Sugestões

1. Leia o enunciado completo, procurando perceber o objectivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer a sua questão.
2. No processo de desenvolvimento do projecto, comece por implementar os vários T.A.I.s pela ordem apresentada no enunciado, seguindo a metodologia respectiva. Em particular, comece por escolher uma representação interna antes de começar a implementar as operações básicas. Só depois deverá desenvolver as funções do jogo, seguindo também a ordem pela qual foram apresentadas no enunciado. Ao desenvolver cada uma das funções pedidas, comece por perceber se pode usar alguma das anteriores.
3. Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste.
4. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.

2 Aspectos a Evitar

Os seguintes aspectos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projecto):

1. Não pense que o projecto 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).
2. Não pense que um dos elementos do seu grupo fará o trabalho por todos. Este é um trabalho de grupo e deverá ser feito em estreita colaboração (comunicação e controle) entre os vários elementos do grupo, cada um dos quais com as suas

responsabilidades. Tanto uma possível oral como perguntas nos testes sobre o projecto, servem para despistar estas situações.

3. *Não duplique código*. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
4. Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.
5. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
6. 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.

3 Classificação

A avaliação da execução será feita através do sistema *Mooshak*. Tal como na primeira parte do projecto, existem vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efectuar uma nova submissão pelo menos 15 minutos depois da submissão anterior. Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos não implica, pois, que esse projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correcto.

Não será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. Os ficheiros de teste usados na avaliação do projecto serão disponibilizados na página da disciplina após a data de entrega.

A nota do projecto será baseada nos seguintes aspectos:

1. **Execução correcta:** 14 valores, distribuídos conforme indicado.

   Esta parte da avaliação é feita recorrendo ao sistema *Mooshak*, que sugere uma nota face aos vários aspectos considerados na avaliação. Para que o programa de avaliação automática consiga avaliar correctamente o seu código, *é fundamental que respeite escrupulosamente as especificações fornecidas no ficheiro anexo.*

2. **Facilidade de leitura:** 4 valores.

   Esta parte da avaliação tem em conta a abstracção procedimental (1 valor), a abstracção de dados (1 valor), nomes bem escolhidos (0.5 valores), paragrafação e indentação correctas (0.5 valores) e a qualidade dos comentários (1 valor). Os seus comentários deverão incluir a assinatura de todas as funções (0.5 valores),

a descrição da representação interna adoptada em cada um dos TAIs definidos (0.5 valores), e outros comentários que entenda relevantes. *Comentários excessivos serão penalizados*.

3. **Estilo de programação:** 2 valores.

4 Condições de Realização e Prazos

A entrega da segunda parte do projecto será efectuada exclusivamente por via electrónica. Deverá submeter um ficheiro `.py` contendo o código do seu projecto através do sistema *Mooshak*, até às **23:59 do dia 20 de Maio de 2014**. Depois desta hora, não serão aceites projectos sob pretexto algum.[5]

O código deverá ser submetido como um único ficheiro com extensão `.py` contendo todo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo.

Importante: Tal como na primeira parte do projecto, no seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII, sob pena de falhar todos os testes. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

Cerca de duas semanas antes do prazo (na Terça-feira, 6 de Maio), serão publicadas na página da cadeira as instruções necessárias para a submissão do código no *Mooshak*. Apenas a partir dessa altura será possível a submissão por via electrónica. Nessa altura serão também fornecidas a cada um as necessárias credenciais de acesso. Para tal, *as inscrições nos grupos deverão ser feitas impreterivelmente até essa data*.

Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação *a última entrega efectuada*. Deverá portanto verificar cuidadosamente que *a última entrega realizada corresponde à versão do projecto que pretende que seja avaliada.* Não serão abertas excepções.[6]

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

Projectos muito semelhantes serão considerados cópia e rejeitados. A detecção de semelhanças entre projectos será realizada utilizando software especializado, e caberá

[5]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 grupos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

[6]Note que, se efectuar uma submissão a menos de 15 minutos do prazo de entrega, fica impossibilitado de efectuar qualquer outra submissão posterior.

exclusivamente ao corpo docente a decisão do que considera ou não cópia. Em caso de cópia, todos os alunos envolvidos terão 0 no projecto e serão reprovados na cadeira.