TÉCNICO LISBOA

Fundamentos da Programação

Ano lectivo 2014-15

Segundo Projecto[1]

2048

2048 é um quebra-cabeças matemático em que um jogador desliza painéis numerados numa grelha quadrada de 4×4 quadrados. O jogo foi criado em Março de 2014 por Gabriele Cirulli, um programador *web* italiano, e está disponível gratuitamente na *web*[2] e como uma *app* para as várias plataformas móveis.

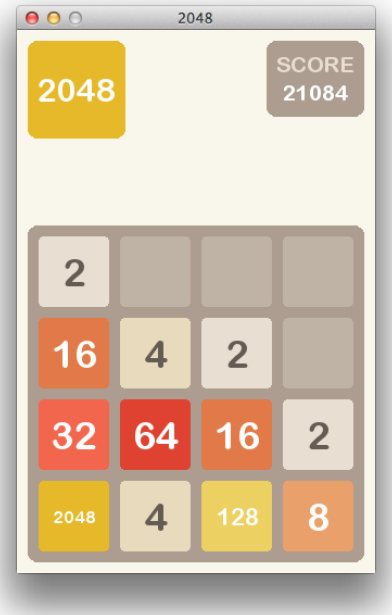


Figura 1: Exemplo de um tabuleiro 2048 com em que foi já criado o bloco 2048.

O objetivo do jogo é combinar painéis com o mesmo número até criar um painel com o número 2048 (ver Fig. 1). Para tal, o jogador pode "arrastar" os painéis em cada uma das quatro direções—para cima (N), baixo (S), esquerda (W) e direita(E). Quando uma ação é aplicada ao tabuleiro, esta afeta *todos* os painéis com espaço para se deslocarem. Por exemplo, a configuração inicial da Fig. 2 poderia dar origem às quatro configurações apresentadas na Fig. 3, consoante a ação aplicada.

[1]Tempo estimado de resolução por uma pessoa, após estudo da matéria correspondente: 30 horas.

[2]Disponível em: `http://gabrielecirulli.github.io/2048/`, acedido pela última vez em 29/10/2014.

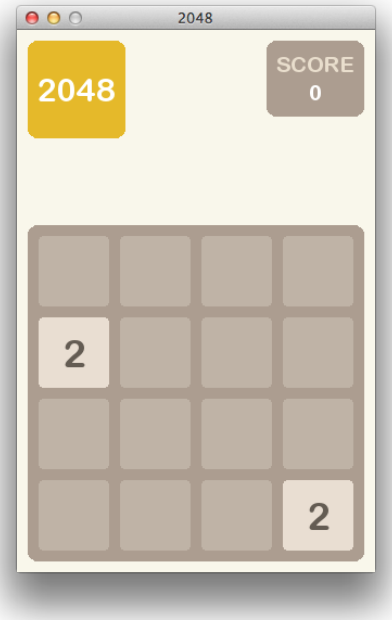
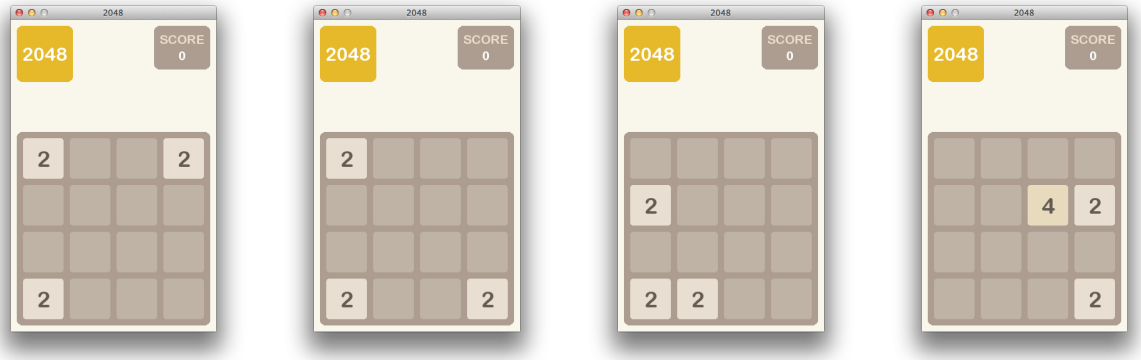


Figura 2: Exemplo de um tabuleiro inicial de 2048.



(a) Acção: N. (b) Acção: S. (c) Acção: W. (d) Acção: E.

Figura 3: Tabuleiro da Fig. 2 após cada uma das quatro ações possíveis.

Após cada movimento, um novo painel aparece numa posição livre do tabuleiro, escolhida aleatoriamente. Este novo painel tem, com uma probabilidade de 0.8, o número 2 e, com uma probabilidade de 0.2, o número 4.

1 Trabalho a Realizar

O objetivo do segundo projeto é escrever um programa em Python que permita a um utilizador jogar o jogo *2048* no computador. Para tal, deverá definir um conjunto de tipos de dados 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.

1.1 Tipos Abstratos de Dados (TAD)

TAD *coordenada* (4 val.)

O TAD *coordenada* será utilizado para indexar as várias posições do tabuleiro. Cada posição do tabuleiro é indexada através da linha respetiva (um inteiro entre 1 e 4) e da coluna respetiva (um inteiro entre 1 e 4), em que a posição $(1, 1)$ corresponde ao canto superior esquerdo do tabuleiro. O TAD *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 TAD 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 4) e o segundo a uma coluna c (um inteiro entre 1 e 4), e deve devolver um elemento do tipo *coordenada* correspondente à posição (l, c). A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem "`cria_coordenada: argumentos invalidos`" caso algum dos argumentos introduzidos seja inválido.

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

  Este seletor recebe como argumento um elemento do tipo *coordenada* e devolve a linha respetiva.

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

  Este seletor recebe como argumento um elemento do tipo *coordenada* e devolve a coluna respetiva.

- *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.

Exemplo de interação:

```
>>> 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
```

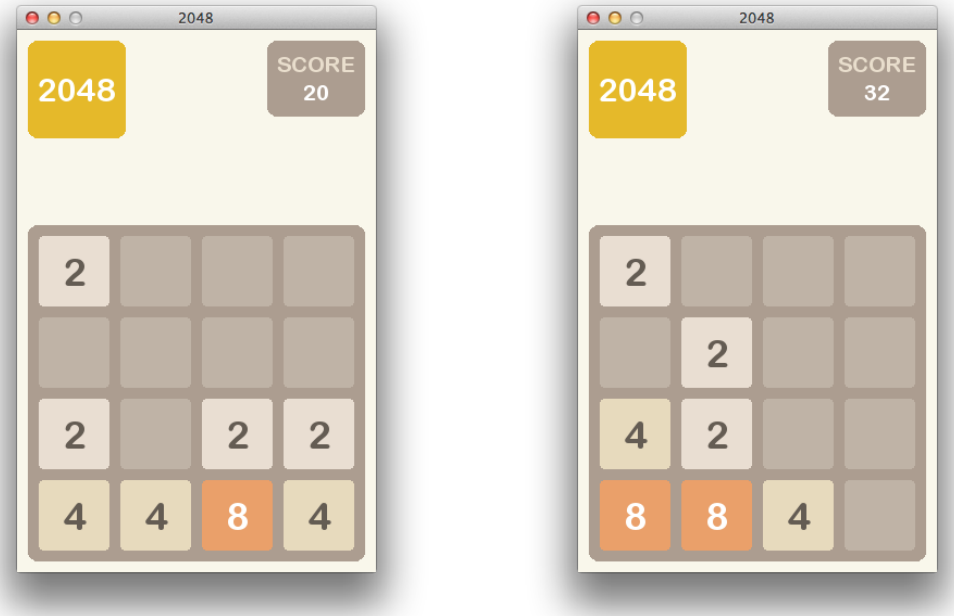
TAD *tabuleiro* (6 val.)

O TAD *tabuleiro* será utilizado para representar o tabuleiro. Este TAD deverá permitir (i) representar um tabuleiro de 2048 (uma grelha de 4×4 posições) e respetiva pontuação; (ii) aceder a cada uma das posições do tabuleiro; (iii) aceder à pontuação do tabuleiro; (iv) modificar o conteúdo de cada uma das posições; (v) actualizar a pontuação do tabuleiro; e (vi) efetuar uma "redução", isto é, mover as peças na grelha numa dada direção e combinar peças contíguas com o mesmo número.

A Fig. 4 apresenta um exemplo de uma "redução" resultante da ação W. Após uma jogada, cada painel efectua o maior deslocamento possível na direcção correspondente até "colidir" com outro painel ou com o extremo do tabuleiro. A ordem pela qual os painéis são deslocados é inversa à direção da jogada. Por exemplo, para a jogada W, os painéis são deslocados em cada linha da direita para a esquerda, *começando pelo painel mais à esquerda e terminando no painel mais à direita*.

Quando um painel colide com outro com o mesmo número, os dois são combinados num único painel cujo valor corresponde ao valor total dos dois painéis originais. Por exemplo, no tabuleiro da Fig. 4, o painel com o valor 4 na posição $(4, 2)$ colide com o painel na posição $(4, 1)$, dando origem a um painel com valor 8 na posição $(4, 1)$.

Um painel resultante da combinação não pode ser combinado com outro painel na mesma jogada. Por exemplo, no tabuleiro da Fig. 4, o painel com o valor 8 na posição $(4, 3)$ colide com o painel de valor 8 na posição $(4, 1)$. No entanto, como este último resultou de uma combinação nesta mesma jogada, a combinação dos dois não é possível.



(a) Tabuleiro inicial. (b) Após acção W.

Figura 4: "Redução" do tabuleiro, resultante de uma ação W. Os painéis com o valor 4 nas penúltima linha são combinados num painel com o valor 8.

O valor de cada painel resultante de uma combinação é somado à pontuação atual.

As operações básicas associadas a este TAD são:

- *cria_tabuleiro* : $\{\} \rightarrow$ *tabuleiro*

  Esta função corresponde ao construtor do tipo *tabuleiro*. Não recebe qualquer argumento, e deverá devolver um elemento do tipo tabuleiro de acordo com a representação interna escolhida. O tabuleiro deverá vir vazio (isto é, todas as posições deverão conter o valor 0 e a pontuação deverá ser 0).

- *tabuleiro_posicao* : *tabuleiro* $\times$ *coordenada* $\rightarrow$ *inteiro*

  Este seletor recebe como argumentos um elemento t do tipo *tabuleiro* e um elemento c do tipo *coordenada* e devolve um elemento do tipo inteiro, que corresponde ao valor na posição do tabuleiro correspondente à coordenada c. Caso a posição correspondente a c esteja vazia, deverá devolver o valor 0. A função deve verificar se o segundo argumento é uma coordenada válida, e gerar um `ValueError` com a mensagem "`tabuleiro_posicao: argumentos invalidos`" caso não seja.

- *tabuleiro_pontuacao* : *tabuleiro* $\rightarrow$*int*

  Este seletor recebe como argumento um elemento t do tipo *tabuleiro* e devolve a pontuação atual para o tabuleiro t.

- *tabuleiro_posicoes_vazias* : *tabuleiro* $\rightarrow$ *lista*

Este seletor recebe como argumento um elemento t do tipo *tabuleiro* devolve uma lista contendo as coordenadas de todas as posições vazias do tabuleiro t.

- *tabuleiro_preenche_posicao* : *tabuleiro* $\times$ *coordenada* $\times$ *inteiro* $\rightarrow$ *tabuleiro*

  Este modificador recebe como argumentos um elemento t do tipo tabuleiro, um elemento c do tipo coordenada e um inteiro v, e *modifica* o tabuleiro t, colocando o valor v na posição correspondente à coordenada c. A função deve devolver o tabuleiro modificado. Deve ainda verificar se c é uma coordenada válida e v um inteiro. Caso algum dos argumentos introduzidos seja inválido deve gerar um `ValueError` com a mensagem "`tabuleiro_preenche_posicao: argumentos invalidos`".

- *tabuleiro_actualiza_pontuacao* : *tabuleiro* $\times$ *inteiro* $\rightarrow$ *tabuleiro*

  Este modificador recebe como argumentos um elemento t do tipo tabuleiro e um inteiro v, não negativo e múltiplo de 4. *Modifica* o tabuleiro t, acrescentando ao valor da respectiva pontuação v pontos. A função deve devolver o tabuleiro modificado. Deve ainda verificar se v é um inteiro não negativo e múltiplo de 4. Caso algum dos argumentos introduzidos seja inválido deve gerar um `ValueError` com a mensagem "`tabuleiro_actualiza_pontuacao: argumentos invalidos`".

- *tabuleiro_reduz* : *tabuleiro* $\times$ *cad. caracteres* $\rightarrow$ *tabuleiro*

  Este modificador recebe como argumento um elemento t do tipo tabuleiro e uma cadeia de caracteres d correspondente a uma das 4 ações possíveis. Em particular d deverá ser uma das cadeias de caracteres `'N'`, `'S'`, `'W'`, `'E'`. A sua função deve modificar o tabuleiro t, reduzindo-o na direção d de acordo com as regras do jogo 2048. A sua função deve devolver o tabuleiro t modificado, incluindo a atualização da pontuação. Deve ainda verificar que d é uma jogada válida (i.e., uma das quatro cadeias de caracteres indicada acima) e gerar um `ValueError` com a mensagem "`tabuleiro_reduz: argumentos invalidos`" caso não seja.

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

  Este reconhecedor recebe um único argumento, devendo devolver `True` se o seu argumento for do tipo *tabuleiro*, e `False` em caso contrário. Não necessita verificar se cada elemento do tabuleiro é uma potência de 2.

- *tabuleiro_terminado* : *tabuleiro* $\rightarrow$ *lógico*

  Este reconhecedor recebe como argumento um elemento t do tipo *tabuleiro* e devolve `True` caso o tabuleiro t esteja terminado, ou seja, caso esteja cheio e não existam movimentos possíveis, e `False` em caso contrário.

- *tabuleiros_iguais* : *tabuleiro* $\times$ *tabuleiro* $\rightarrow$ *lógico*

  Este teste recebe como argumentos dois elementos t_1 e t_2 do tipo *tabuleiro* e devolve `True` caso t_1 e t_2 correspondam a dois tabuleiros com a mesma configura-

ção e pontuação, e `False` em caso contrário.

Deve ainda implementar o seguinte transformador de saída:

- *escreve_tabuleiro* : *tabuleiro* $\rightarrow$ {}

  A função *escreve_tabuleiro* recebe como argumento um elemento t do tipo *tabuleiro* e escreve para o ecrã a representação externa de um tabuleiro de 2048, apresentada no exemplo abaixo. Deve ainda verificar se t é um tabuleiro válido e, caso o argumento introduzido seja inválido, deve gerar um `ValueError` com a mensagem "`escreve_tabuleiro: argumentos invalidos`".

Exemplo de interação:

```
>>> t = cria_tabuleiro(0)
>>> escreve_tabuleiro(t)
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 0
>>> c = cria_coordenada(2, 2)
>>> escreve_tabuleiro(tabuleiro_preenche_posicao(t, c, 2))
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 2 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 0
>>> escreve_tabuleiro(t)
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 2 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 0
>>> tabuleiro_posicao(t, 0)
Traceback (most recent call last):
[...]
builtins.ValueError: tabuleiro_posicao: argumentos invalidos
>>> tabuleiro_posicao(t, cria_coordenada(2, 4))
0
>>> tabuleiro_posicao(t, cria_coordenada(2, 2))
2
>>> c = cria_coordenada(2, 4)
>>> escreve_tabuleiro(tabuleiro_preenche_posicao(t, c, 2))
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 2 ] [ 0 ] [ 2 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 0
>>> cria_coordenada(1, 1) in tabuleiro_posicoes_vazias(t)
True
>>> cria_coordenada(2, 2) in tabuleiro_posicoes_vazias(t)
False
```

```
>>> e_tabuleiro(t)
True
>>> e_tabuleiro('a')
False
>>> escreve_tabuleiro(tabuleiro_reduz(t, 'E'))
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 4 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 4
>>> escreve_tabuleiro(t)
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 4 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
[ 0 ] [ 0 ] [ 0 ] [ 0 ]
Pontuacao: 4
```

1.2 Funções Adicionais

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

  Esta função não recebe qualquer argumento, limitando-se a pedir ao utilizador para introduzir uma direção (N, S, E ou W). Caso o valor introduzidos pelo utilizador seja inválido, a função deve pedir novamente a informação de jogada ao utilizador. A função devolve uma cadeia de caracteres correspondente à direção escolhida pelo utilizador.

  Exemplo:

```
>>> j = pede_jogada()
Introduza uma jogada (N, S, E, W): 1
Jogada invalida.
Introduza uma jogada (N, S, E, W): a
Jogada invalida.
Introduza uma jogada (N, S, E, W): n
Jogada invalida.
Introduza uma jogada (N, S, E, W): N
>>> j
'N'
```

- *jogo_2048* : $\{\} \rightarrow \{\}$ **(3 val.)**

  Esta função corresponde à função principal do jogo. Não recebe qualquer argumento, e permite a um utilizador jogar um jogo completo de 2048.

  Em cada turno, a função deve escrever o tabuleiro no ecrã e pedir ao utilizador para introduzir uma nova jogada. Caso a jogada seja válida, deverá atualizar 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.

Sugestão: Poderá ainda ser útil implementar duas funções adicionais, nomeadamente `copia_tabuleiro` e `preenche_posicao_aleatoria`. A primeira recebe como argumento um elemento do tipo tabuleiro e devolve uma cópia do mesmo. A segunda recebe um elemento do tipo tabuleiro e preenche uma posição livre, escolhida aleatoriamente, com um dos números 2 ou 4, de acordo com as probabilidades já indicadas. Estas sugestões, no entanto, não são obrigatórias.

1.3 Sugestões

1. Leia o enunciado completo, procurando perceber o objetivo 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 projeto, comece por implementar os vários T.A.D.s pela ordem apresentada no enunciado, seguindo a metodologia respetiva. 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 Aspetos a Evitar

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

1. 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).
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 projeto, servem para despistar estas situações.

3. Pode discutir com colegas de outros grupos ideias para a implementação do código, mas não copie partes do código, mesmo que modificadas. Deve compreender totalmente cada componente do seu código, o que é muito difícil quando partes do código são copiadas de (ou inspiradas em) código escrito por outras pessoas.
4. *Não duplique código*. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
5. Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.
6. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
7. 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 projeto, existem vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efetuar 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 projeto completar com sucesso os exemplos fornecidos não implica, pois, que esse projeto esteja totalmente correto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada grupo garantir que o código produzido está correto.

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 projeto serão disponibilizados na página da disciplina após a data de entrega.

A nota do projeto será baseada nos seguintes aspetos:

1. Execução correta (70%).

   Esta parte da avaliação é feita recorrendo ao sistema *Mooshak* que sugere uma nota face aos vários aspetos considerados.

2. Facilidade de leitura, nomeadamente a abstração procedimental, a abstração de dados, nomes bem escolhidos, qualidade (e não quantidade) dos comentários e tamanho das funções (25%). Os seus comentários deverão incluir, entre outros, uma descrição da representação interna adotada em cada um dos TAIs definidos.

3. Estilo de programação (5%).

4 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 **23:59 do dia 10 de Dezembro de 2014**. Depois desta hora, não serão aceites projetos sob pretexto algum.[3]

Deverá submeter um único ficheiro com extensão `.py` contendo todo o código do seu projeto. 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.

No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

Duas semanas antes do prazo, 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 eletrónica. Nessa altura serão também fornecidas a cada um as necessárias credenciais de acesso. Até ao prazo de entrega poderá efetuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efetuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada. Não serão abertas exceções.*

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

Projetos iguais, ou muito semelhantes, serão penalizados com a reprovação na disciplina. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

[3]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.