TÉCNICO LISBOA

Fundamentos da Programação

Ano lectivo 2013-14, 2º Semestre

Projecto – Primeira Parte[1]

Nim

Nim é um jogo de estratégia matemática, que pode ser jogado casualmente usando um conjunto de objectos dispostos em pilhas (e.g., fósforos). O nome do jogo foi proposto pelo matemático Charles Bouton, da Universidade de Harvard, que estudou exaustivamente as propriedades do mesmo [1], embora a origem jogo remonte a tempos bem mais antigos.[2]

No jogo *Nim* dois jogadores vão alternadamente removendo objectos de um conjunto de pilhas. Em cada jogada, o jogador respectivo tem que remover no mínimo um objecto, não havendo limite máximo em relação ao número de objectos removidos *desde que sejam todos da mesma pilha*. Ganha o jogador que remover o último objecto.[3]

1 Trabalho a Realizar

O objectivo da primeira parte do projecto é escrever um programa em Python que permita jogar a versão tradicional do jogo *Nim* no computador. Para tal, o seu programa deverá armazenar o estado do jogo em cada jogada (correspondendo este simplesmente ao número de objectos em cada uma das três pilhas, conforme Fig. 1), e ir alternadamente solicitando a cada jogador que efectue uma jogada.

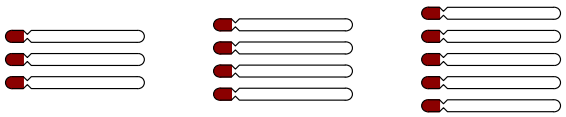

Figura 1: Exemplo de uma configuração possível para o jogo *Nim*.

[1] Tempo estimado de resolução por um grupo de 3 pessoas, após estudo da matéria correspondente: 6 horas.

[2] As primeiras referências a este jogo na Europa datam do início do séc. XVI.

[3] Há inúmeras variantes do jogo, sendo outra igualmente conhecida a versão *misère*, em que o jogador que retira o último objecto é o perdedor.

No seu projecto, deverá implementar as seguintes funções (a cotação de cada função encontra-se indicada entre parêntesis).

- *pede_jogada* : $int \times int \times int \times int \rightarrow int \times int$ **(4 valores)**

  Esta função recebe como argumentos 4 inteiros, os três primeiros correspondendo ao número de objectos em cada uma das 3 pilhas e o último correspondente ao número do jogador (sendo este 1 ou 2). O número de objectos em cada pilha tem que ser um inteiro não-negativo e menor que 10.

  A função deve escrever para o ecrã o número do jogador ao qual se vai pedir uma jogada e, de seguida, pedir ao utilizador para (i) escolher de que pilha pretende retirar objectos; e (ii) quantos objectos pretende retirar da pilha. Caso o utilizador introduza informação inválida, a função deve pedir ao utilizador para repetir o processo.

  A função devolve dois inteiros, correspondentes ao número da pilha e ao número de objectos escolhidos pelo utilizador, respectivamente.

  Exemplo:

```
>>> pede_jogada(1, 2, 3, 0)
[...]
builtins.ValueError: pede_jogada: numero de jogador invalido
>>> pede_jogada(-1, 2, 3, 0)
[...]
builtins.ValueError: pede_jogada: valor invalido nas pilhas
>>> pede_jogada(-1, 2, 3, 1)
[...]
builtins.ValueError: pede_jogada: valor invalido nas pilhas
>>> pede_jogada(1, 2, 3, 1)
- Jogador 1 -
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 1): 2
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 1): 0
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 1): 1
(1, 1)
```

- *desenha_jogo* : $int \times int \times int \rightarrow \{\}$ **(2 val.)**

  Esta função recebe como argumentos 3 inteiros, correspondendo ao número de objectos em cada uma das 3 pilhas, e deve desenhar para no ecrã uma representação gráfica do mesmo, como ilustrado no exemplo abaixo.

  A representação gráfica é constituída por tantas linhas como o número de objectos na pilha maior, seguidas de uma linha indicando o número de objectos em cada uma das 3 pilhas. Os objectos/espaços das várias pilhas estão separadas horizontalmente por 3 espaços.

Exemplo:

```
>>> desenha_jogo(-1, 1, 1)
[...]
builtins.ValueError: desenha_jogo: valor invalido nas pilhas
>>>desenha_jogo(1, 1, 1)

o-----   o-----   o-----

p1 = 1   p2 = 1   p3 = 1
>>>desenha_jogo(2, 3, 4)

                  o-----
         o-----   o-----
o-----   o-----   o-----
o-----   o-----   o-----

p1 = 2   p2 = 3   p3 = 4
```

Para efeitos de clarificação, apresenta-se o mesmo exemplo, mas desta vez assinalando os espaços com o símbolo "␣" e as mudanças de linha com o símbolo "¶".

```
>>> desenha_jogo(-1, 1, 1)
[...]
builtins.ValueError: desenha_jogo: valor invalido nas pilhas
>>>desenha_jogo(1, 1, 1)
¶
o-----␣␣␣o-----␣␣␣o-----¶
¶
p1␣=␣1␣␣␣p2␣=␣1␣␣␣p3␣=␣1¶
>>>desenha_jogo(2, 3, 4)
¶
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣o-----¶
␣␣␣␣␣␣␣␣␣o-----␣␣␣o-----¶
o-----␣␣␣o-----␣␣␣o-----¶
o-----␣␣␣o-----␣␣␣o-----¶
¶
p1␣=␣2␣␣␣p2␣=␣3␣␣␣p3␣=␣4¶
```

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

  Esta função recebe como argumentos 3 inteiros, correspondendo ao número de objectos em cada uma das 3 pilhas. A função deve devolver `True` caso o número de objectos nas pilhas corresponda a um jogo terminado, e `False` em caso contrário.

  Exemplo:

```
>>> jogo_terminado(-1, 1, 1)
[...]
builtins.ValueError: jogo_terminado: valor invalido nas pilhas
>>> jogo_terminado(1, 2, 3)
False
```

```
>>> jogo_terminado(0, 0, 0)
True
```

- *actualiza_pilhas* : $int \times int \times int \times int \times int \rightarrow int \times int \times int$ **(2 val.)**

  Esta função recebe como argumentos 5 inteiros, os três primeiros correspondendo ao número de objectos em cada uma das 3 pilhas, o quarto correspondendo ao número de uma pilha, p (um inteiro entre 1 e 3), e o último correspondendo a um número de objectos, o. O par (p, o) representa a jogada correspondente a retirar o objectos da pilha p.

  A função deve devolver 3 inteiros, correspondentes ao número de objectos em cada uma das 3 pilhas *após* a realização da jogada (p, o).

  Exemplo:

```
>>> p1, p2, p3 = actualiza_pilhas(-1, 1, 1, 1, 1)
[...]
builtins.ValueError: actualiza_pilhas: valor invalido nas pilhas
>>> p1, p2, p3 = actualiza_pilhas(1, 1, 1, -1, 1)
[...]
builtins.ValueError: actualiza_pilhas: pilha invalida
>>> p1, p2, p3 = actualiza_pilhas(1, 1, 1, 1, 2)
[...]
builtins.ValueError: actualiza_pilhas: numero de objectos invalido
>>> p1, p2, p3 = actualiza_pilhas(1, 1, 1, 1, 1)
>>> p1
0
>>> p2
1
>>> p3
1
```

- *jogo_nim* : $int \times int \times int \times logico \rightarrow \{\}$ **(5 val.)**

  Esta função recebe como argumentos 3 inteiros, correspondendo ao número de objectos em cada uma das 3 pilhas, e um valor lógico, que indica se o jogo terá ou não lugar com um jogador automático. Caso o último argumento corresponda ao valor lógico `True`, o segundo jogador corresponderá a um jogador automático fornecido pelo corpo docente.

  A função deverá pedir alternadamente aos dois jogadores para jogar e, quando relevante, anunciar o o resultado, escrevendo no ecrã:

  - "`Perdeu o jogador 1. O vencedor foi o jogador automatico.`" no caso em que o jogador automático vença o jogador humano.
  - "`Parabens, ganhou ao jogador automatico.`" no caso em que o jogador humano vença o jogador automático.
  - "`Perdeu o jogador 1. O vencedor foi o jogador 2.`" no caso em que o segundo jogador humano vença o primeiro jogador humano.

- "`Perdeu o jogador 2. O vencedor foi o jogador 1.`" no caso em que o primeiro jogador humano vença o segundo jogador humano.

Para ter acesso ao jogador automático no seu programa deverá incluir, no início do seu ficheiro de código, a seguinte instrução:

```
from auxiliar import *
```

Deve ainda assegurar-se que o ficheiro `auxiliar.pyc` fornecido pelo corpo docente se encontra na mesma directoria que o seu ficheiro de código. Para pedir uma jogada ao jogador automático, deverá usar a função `automatico`, que recebe como argumentos 3 inteiros (correspondendo ao número de objectos em cada uma das 3 pilhas) e, tal como a função `pede_jogada`, devolve dois inteiros, correspondentes ao número da pilha e ao número de objectos escolhidos pelo jogador automático, respectivamente.

A função `automatico` já escreve a informação necessária para o ecrã.

Exemplo:

```
>>> p, o = automatico(2, 1, 1)
- Jogador 2 -
O automatico vai retirar 2 objectos da pilha 1
>>> p
1
>>> o
2
```

Exemplo de um jogo *Nim* com jogador automático:

```
>>> jogo_nim(1, 2, 3, True)

                   o-----
          o-----   o-----
o-----    o-----   o-----

p1 = 1   p2 = 2   p3 = 3
- Jogador 1 -
Seleccione uma pilha (1, 2 ou 3): 3
Indique o numero de objectos a retirar (max: 3): 1

          o-----   o-----
o-----    o-----   o-----

p1 = 1   p2 = 2   p3 = 2
- Jogador 2 -
O automatico vai retirar 1 objecto da pilha 1

          o-----   o-----
          o-----   o-----

p1 = 0   p2 = 2   p3 = 2
- Jogador 1 -
Seleccione uma pilha (1, 2 ou 3): 2
Indique o numero de objectos a retirar (max: 2): 3
```

```
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 0): 2
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 0): 0
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 2
Indique o numero de objectos a retirar (max: 2): 2

                  o-----
                  o-----

p1 = 0   p2 = 0   p3 = 2
- Jogador 2 -
O automatico vai retirar 2 objectos da pilha 3
Perdeu o jogador 1. O vencedor foi o jogador automatico.
```

Exemplo de um jogo *Nim* com dois jogadores humanos:

```
>>> jogo_nim(1, 1, 1, False)

o-----   o-----   o-----

p1 = 1   p2 = 1   p3 = 1
- Jogador 1 -
Seleccione uma pilha (1, 2 ou 3): 4
Pilha invalida.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 1): 1

         o-----   o-----

p1 = 0   p2 = 1   p3 = 1
- Jogador 2 -
Seleccione uma pilha (1, 2 ou 3): 2
Indique o numero de objectos a retirar (max: 1): 2
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 1
Indique o numero de objectos a retirar (max: 0): 0
Numero de objectos invalido.
Seleccione uma pilha (1, 2 ou 3): 2
Indique o numero de objectos a retirar (max: 1): 1

                  o-----

p1 = 0   p2 = 0   p3 = 1
- Jogador 1 -
Seleccione uma pilha (1, 2 ou 3): 3
Indique o numero de objectos a retirar (max: 1): 1
Perdeu o jogador 2. O vencedor foi o jogador 1.
```

Todas as funções devem validar os argumentos pela ordem porque aparecem. Em particular, todas as funções devem verificar que o valor de cada uma das pilhas é um inteiro

não-negativo menor que 10. Além disso,

- A função `pede_jogada` deve verificar que o último argumento é 1 ou 2.
- A função `actualiza_pilhas` deve verificar que os dois últimos argumentos correspondem a uma jogada válida, isto é, que o quarto argumento corresponde a um número de pilha válido, e que o quinto argumento corresponde a um número válido de objectos, dada a pilha.
- A função `jogo_nim` deve verificar que o último argumento é do tipo lógico.

Sempre que uma função receba um argumento inválido (de acordo com as verificações acima), deverá gerar um *erro de valor* (`ValueError`) com a mensagem de erro adequada, de acordo com os exemplos fornecidos.

2 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. Procure desenvolver/implementar as funções pela 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.

3 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.

4 Classificação

A avaliação da execução será feita através de um sistema automático para entrega de projectos designado *Mooshak*. Será demonstrada numa das aulas teóricas o funcionamento com o sistema *Mooshak*. 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 no enunciado, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos no enunciado 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.

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), nomes bem escolhidos (0.5 valores), paragrafação e indentação correctas (0.5 valores) e a qualidade dos comentários (2 valores). Os seus comentários deverão incluir a assinatura de todas as funções (1.5 valores), bem como outros comentários que entenda relevantes (0.5 valores). *Comentários excessivos serão penalizados*.

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

5 Condições de Realização e Prazos

A entrega da primeira parte do projecto será efectuada exclusivamente por via electrónica. Deverá submeter o seu projecto através do sistema *Mooshak*, até às **23:59 do dia 25 de Março de 2013**. Depois desta hora, não serão aceites projectos sob pretexto algum.[4]

Deverá submeter 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: 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 (isto é, na Segunda-feira, 14 de Março), 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.[5]

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 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.

Referências

[1] C. L. Bouton. *Nim*, a game with a complete mathematical theory. *Annals of Mathematics*, 3:35–39, 1901.

[4] Note que o limite de 10 submissões simultâneas 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.

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