TÉCNICO LISBOA

Fundamentos da Programação
Ano lectivo 2020-21
Primeiro Projecto
6 de Novembro de 2020

Jogo do Galo

Neste primeiro projecto de Fundamentos da Programação os alunos irão desenvolver as funções de forma a implementar um programa em Python que permita a um jogador humano jogar o Jogo do Galo contra o computador.

1 Descrição do jogo

1.1 Regras do jogo

O jogo do galo é um jogo muito popular de *papel e lápis* para dois jogadores, jogando com os símbolos 'X' e 'O' de forma alternada nas posições de um tabuleiro de dimensão 3x3. O jogador que conseguir colocar três dos seus símbolos numa linha horizontal, vertical ou diagonal é o vencedor. Se não haver nenhum vencedor após se terem marcado todas as posições possíveis, o jogo termina num empate.

1.2 Estratégia de jogo

O jogo do galo é um jogo com uma combinatória simples que permite desenvolver soluções eficientes para a escolha do melhor movimento baseada em algoritmos de teoria de jogos[1]. Alternativamente, e conforme o usado por Newell e Simon's em 1972[2], os jogadores podem realizar um jogo perfeito (para ganhar ou pelo menos empatar) se, cada vez que for sua vez de jogar, escolherem a primeira acção disponível da lista a seguir:

1. **Vitória:**
   Se o jogador tiver duas das suas peças em linha e uma posição livre
   então deve marcar na posição livre (ganhando o jogo);

2. **Bloqueio:**
   Se o adversário tiver duas das suas peças em linha e uma posição livre
   então deve marcar na posição livre (para bloquear o adversário);

3. **Bifurcação:**
   Se o jogador tiver duas linhas/colunas/diagonais que se intersectam, onde cada uma contém uma das suas peças
   e se a posição de intersecção estiver livre
   então deve marcar na posição de intersecção (criando duas formas de vencer na jogada seguinte);

[1] https://en.wikipedia.org/wiki/Minimax
[2] https://onlinelibrary.wiley.com/doi/epdf/10.1207/s15516709cog1704_3

4. **Bloqueio de bifurcação:**
   Se o adversário tiver apenas uma bifurcação
   então o jogador deve bloquear a bifurcação (escolher a posição livre da intersecção)
   senão o jogador deve criar um dois em linha para forçar o oponente a defender, desde que a defesa não resulte na criação de uma bifurcação para o oponente;

5. **Centro:**
   Se a posição central estiver livre
   então jogar na posição central;

6. **Canto oposto:**
   Se o adversário estiver num canto
   e se o canto diagonalmente oposto for uma posição livre
   então jogar nesse canto oposto;

7. **Canto vazio:**
   Se um canto for uma posição livre
   então jogar nesse canto;

8. **Lateral vazio:**
   Se uma posição lateral (que nem é o centro, nem um canto) for livre
   então jogar nesse lateral.

2 Trabalho a realizar

O objetivo do primeiro projecto é escrever um programa em Python, correspondendo às funções descritas nesta secção, que permita jogar ao Jogo do Galo conforme descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso considere necessário.

2.1 Representação do tabuleiro

Considere que um *tabuleiro* de dimensão 3x3 é representado internamente (ou seja, no seu programa) por um tuplo com 3 tuplos, cada um deles contendo 3 valores inteiros. Os valores representam cada uma das posições do tabuleiro podendo tomar valores iguais a 1, -1 ou 0, dependendo se a posição tiver sido marcada com os símbolos 'X', 'O' ou se estiver livre, respectivamente. Considere também que cada uma das posições dum tabuleiro é representada internamente por um número inteiro seguindo a ordem da esquerda para a direita e de cima para baixo como ilustrado na Figura 1.
As funções a escrever durante o projecto são as seguintes:

2.1.1 eh_tabuleiro: universal → booleano (1.25 valores)

Esta função recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a um tabuleiro e `False` caso contrário, sem nunca gerar erros.

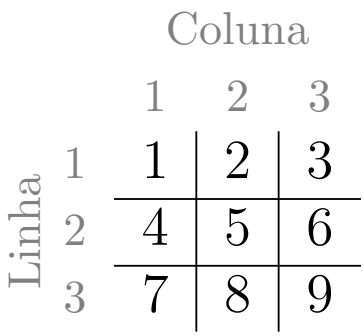

Figura 1: Posições dum tabuleiro do jogo do galo.

```
>>> tab = ((1,0,0),(-1,1,0),(1,-1,-1))
>>> eh_tabuleiro(tab)
True
>>> tab = ((1,0,0),('O',1,0),(1,-1,-1))
>>> eh_tabuleiro(tab)
False
>>> tab = ((1,0,0),(-1,1,0),(1,-1))
>>> eh_tabuleiro(tab)
False
```

2.1.2 eh_posicao: universal → booleano (0.5 valores)

Esta função recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a uma posição e `False` caso contrário, sem nunca gerar erros. Considere que as posições no tabuleiro são definidas por um inteiro, como indicado na Figura 1.

```
>>> eh_posicao(9)
True
>>> eh_posicao(-2)
False
>>> eh_posicao((1,))
False
```

2.1.3 obter_coluna: tabuleiro ×inteiro → vector (0.75 valores)

Esta função recebe um tabuleiro e um inteiro com valor de 1 a 3 que representa o número da coluna, e devolve um vector com os valores dessa coluna. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'obter_coluna: algum dos argumentos e invalido'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> obter_coluna(tab, 2)
(-1, 0, -1)
>>> obter_coluna(tab, 1)
(1, 1, 1)
>>> obter_coluna(tab, 4)
Traceback (most recent call last): <...>
ValueError: obter_coluna: algum dos argumentos e invalido
```

2.1.4 obter_linha: tabuleiro ×inteiro → vector (0.5 valores)

Esta função recebe um tabuleiro e um inteiro com valor de 1 a 3 que representa o número da linha, e devolve um vector com os valores dessa linha. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'obter_linha: algum dos argumentos e invalido'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> obter_linha(tab, 2)
(1, 0, -1)
>>> obter_linha(tab, 1)
(1, -1, 0)
>>> obter_linha(tab, 4)
Traceback (most recent call last): <...>
ValueError: obter_linha: algum dos argumentos e invalido
```

2.1.5 obter_diagonal: tabuleiro ×inteiro → vector (0.75 valores)

Esta função recebe um tabuleiro e um inteiro que representa a direcção da diagonal, 1 para descendente da esquerda para a direita e 2 para ascendente da esquerda para a direita, e devolve um vector com os valores dessa diagonal. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'obter_diagonal: algum dos argumentos e invalido'`.

```
>>> tab = ((1,-1,-1),(1,0,-1),(1,-1,0))
>>> obter_diagonal(tab, 1)
(1, 0, 0)
>>> obter_diagonal(tab, 2)
(1, 0, -1)
>>> obter_diagonal(tab, 3)
Traceback (most recent call last): <...>
ValueError: obter_diagonal: algum dos argumentos e invalido
```

2.1.6 tabuleiro_str: tabuleiro → cad. carateres (1.25 valores)

Esta função recebe um tabuleiro e devolve a cadeia de caracteres que o representa (a representação externa ou representação "para os nossos olhos"), de acordo com o exemplo na seguinte interacção. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'tabuleiro_str: o argumento e invalido'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> tabuleiro_str(tab)
' X | O |   \n-----------\n X |   | O \n-----------\n X | O |   '
>>> print(tabuleiro_str(tab))
 X | O |
-----------
 X |   | O
-----------
 X | O |
>>> tabuleiro_str(((1,-1,0),(1,0,-1)))
Traceback (most recent call last): <...>
ValueError: tabuleiro_str: o argumento e invalido
```

2.2 Funções de inspecção e manipulação do tabuleiro

2.2.1 eh_posicao_livre: tabuleiro ×posicao → booleano (0.75 valores)

Esta função recebe um tabuleiro e uma posição, e devolve `True` se a posição corresponde a uma posição livre do tabuleiro e `False` caso contrário. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'eh_posicao_livre: algum dos argumentos e invalido'`.

```
>>> tab = ((1,-1,0),(1,-1,0),(1,-1,0))
>>> eh_posicao_livre(tab, 9)
True
>>> eh_posicao_livre(tab, 7)
False
>>> eh_posicao_livre(tab, (-1,))
Traceback (most recent call last): <...>
ValueError: eh_posicao_livre: algum dos argumentos e invalido
```

2.2.2 obter_posicoes_livres: tabuleiro → vector (0.75 valores)

Esta função recebe um tabuleiro, e devolve o vector ordenado com todas as posições livres do tabuleiro. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'obter_posicoes_livres: o argumento e invalido'`.

```
>>> tab = ((1,-1,0),(1,-1,0),(1,-1,0))
>>> obter_posicoes_livres(tab)
(3, 6, 9)
>>> tab = ((1,-1,0),(1,-1,0),(1,-1))
>>> obter_posicoes_livres(tab)
Traceback (most recent call last): <...>
ValueError: obter_posicoes_livres: o argumento e invalido
```

2.2.3 jogador_ganhador: tabuleiro → inteiro (2 valores)

Esta função recebe um tabuleiro, e devolve um valor inteiro a indicar o jogador que ganhou a partida no tabuleiro passado por argumento, sendo o valor igual a 1 se ganhou o jogador que joga com 'X', -1 se ganhou o jogador que joga com 'O', ou 0 se não ganhou nenhum jogador. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'jogador_ganhador: o argumento e invalido'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(0,-1,0))
>>> jogador_ganhador(tab)
0
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> jogador_ganhador(tab)
1
>>> tab = ((1,1,-1),(-1,-1,1),(-1,-1,1))
>>> jogador_ganhador(tab)
-1
>>> tab = ((1,1,-1),(-1,-1,1),(1,-1))
>>> jogador_ganhador(tab)
Traceback (most recent call last): <...>
ValueError: jogador_ganhador: o argumento e invalido
```

2.2.4 marcar_posicao: tabuleiro ×inteiro ×posicao → tabuleiro (1 valor)

Esta função recebe um tabuleiro, um inteiro identificando um jogador (1 para o jogador 'X' ou -1 para o jogador 'O') e uma posição *livre*, e devolve um novo tabuleiro modificado com uma nova marca do jogador nessa posição. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'marcar_posicao: algum dos argumentos e invalido'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> marcar_posicao(tab, -1, 5)
((1, -1, 0), (1, -1, -1), (1, -1, 0))
>>> marcar_posicao(tab, 1, 3)
```

```
((1, -1, 1), (1, 0, -1), (1, -1, 0))
>>> marcar_posicao(tab, 1, 1)
Traceback (most recent call last): <...>
ValueError: marcar_posicao: algum dos argumentos e invalido
```

2.3 Funções de jogo

2.3.1 escolher_posicao_manual: tabuleiro → posicao (0.5 valores)

Esta função realiza a leitura de uma posição introduzida manualmente por um jogador e devolve esta posição escolhida. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'escolher_posicao_manual: o argumento e invalido'`. A função deve apresentar a mensagem `'Turno do jogador. Escolha uma posicao livre: '`, para pedir ao utilizador para introduzir uma posição livre. Pode assumir que o utilizador introduzirá sempre um número inteiro. Se o valor introduzido não corresponder a uma posição livre do tabuleiro, a função deve gerar um erro com a mensagem `'escolher_posicao_manual: a posicao escolhida e invalida'`.

```
>>> tab = ((1,-1,0),(1,0,-1),(1,-1,0))
>>> escolher_posicao_manual(tab)
Turno do jogador. Escolha uma posicao livre: 3
3
>>> escolher_posicao_manual(tab)
Turno do jogador. Escolha uma posicao livre: 12
Traceback (most recent call last): <...>
ValueError: escolher_posicao_manual: a posicao introduzida e invalida
```

2.3.2 escolher_posicao_auto: tabuleiro ×inteiro ×cad. carateres → posicao (4 valores)

Esta função recebe um tabuleiro, um inteiro identificando um jogador (1 para o jogador 'X' ou -1 para o jogador 'O') e uma cadeia de carateres correspondente à estratégia, e devolve a posição escolhida automaticamente de acordo com a estratégia seleccionada. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'escolher_posicao_auto: algum dos argumentos e invalido'`.
As estratégias a seguir devem ser as seguintes:

- **'basico'** (1 valor) - deve considerar em ordem os critérios 5, 7 e 8 descritos nas secção 1.2;
- **'normal'** (1.5 valores) - deve considerar em ordem os critérios 1, 2, 5, 6, 7 e 8 descritos nas secção 1.2;

- **'perfeito'** (1.5 valores) - deve seguir uma abordagem perfeita, de acordo com a estratégia completa descrita na secção 1.2.

Sempre que houver mais do que uma posição que cumpra um dos critérios definidos nas estratégias anteriores, deve escolher a primeira posição seguindo a numeração das posições definida anteriormente na Figura 1 (da esquerda para a direita e de cima para baixo).

```
>>> tab = ((0,0,0),(0,1,0),(-1,0,0))
>>> escolher_posicao_auto(tab, 1, 'basico')
1
>>> escolher_posicao_auto(tab, 1, 'normal')
3
>>> tab = ((0,0,-1),(-1,1,0),(1,0,0))
>>> escolher_posicao_auto(tab, 1, 'normal')
1
>>> escolher_posicao_auto(tab, 1, 'perfeito')
8
>>> escolher_posicao_auto(tab, 'O', 'basico')
Traceback (most recent call last): <...>
ValueError: escolher_posicao_auto: algum dos argumentos e invalido
```

2.3.3 jogo_do_galo: cad. carateres ×cad. carateres → cad. carateres (2 valores)

Esta função corresponde à função principal que permite jogar um jogo completo de Jogo do Galo de um jogador contra o computador. O jogo começa sempre com o jogador 'X' a marcar uma posição livre e termina quando um dos jogadores vence ou, se não existem posições livres no tabuleiro. A função recebe duas cadeias de caracteres e devolve o identificador do jogador ganhador ('X' ou 'O'). Em caso de empate, a função deve devolver a cadeia de caracteres 'EMPATE'. O primeiro argumento corresponde a marca ('X' ou 'O') que deseja utilizar o jogador humano, e o segundo argumento selecciona a estratégia de jogo utilizada pela máquina. Se algum dos argumentos dados forem inválidos, a função deve gerar um erro com a mensagem `'jogo_do_galo: algum dos argumentos e invalido'`. A função deve apresentar a mensagem `'Turno do computador (<estrategia>):'`, em que `<estrategia>` corresponde à cadeia de caracteres passada como argumento, quando for o turno do computador.

Exemplo 1

```
>>> jogo_do_galo('O', 'basico')
Bem-vindo ao JOGO DO GALO.
O jogador joga com 'O'.
Turno do computador (basico):
   |   |
-----------
   | X |
-----------
   |   |
Turno do jogador. Escolha uma posicao livre: 1
 O |   |
-----------
   | X |
-----------
   |   |
Turno do computador (basico):
 O |   | X
-----------
   | X |
-----------
   |   |
Turno do jogador. Escolha uma posicao livre: 7
 O |   | X
-----------
   | X |
-----------
 O |   |
Turno do computador (basico):
 O |   | X
-----------
   | X |
-----------
 O |   | X
Turno do jogador. Escolha uma posicao livre: 4
 O |   | X
-----------
 O | X |
-----------
 O |   | X
'O'
>>>
```

Exemplo 2

```
>>> jogo_do_galo('X', 'perfeito')
Bem-vindo ao JOGO DO GALO.
O jogador joga com 'X'.
Turno do jogador. Escolha uma posicao livre: 1
 X |   |
-----------
   |   |
-----------
   |   |
Turno do computador (perfeito):
 X |   |
-----------
   | O |
-----------
   |   |
Turno do jogador. Escolha uma posicao livre: 9
 X |   |
-----------
   | O |
-----------
   |   | X
Turno do computador (perfeito):
 X | O |
-----------
   | O |
-----------
   |   | X
Turno do jogador. Escolha uma posicao livre: 8
 X | O |
-----------
   | O |
-----------
   | X | X
Turno do computador (perfeito):
 X | O |
-----------
   | O |
-----------
 O | X | X
Turno do jogador. Escolha uma posicao livre: 3
 X | O | X
-----------
```

```
   | O |
-----------
 O | X | X
Turno do computador (perfeito):
 X | O | X
-----------
   | O | O
-----------
 O | X | X
Turno do jogador. Escolha uma posicao livre: 4
 X | O | X
-----------
 X | O | O
-----------
 O | X | X
'EMPATE'
>>>
```

3 Condições de Realização e Prazos

- A entrega do 1º projecto será efetuada exclusivamente por via eletrónica. Deverá submeter o seu projecto através do sistema Mooshak, até às **17:00 do dia 23 de Novembro de 2020**. Depois desta hora, não serão aceites projectos sob pretexto algum.

- Deverá submeter um único ficheiro com extensão *.py* contendo todo o código do seu projecto. **O ficheiro de código deverá conter em comentário, na primeira linha, o número e o nome do aluno**.

- No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer outro carácter não pertencente à tabela ASCII. Todos os testes automáticos falharão, mesmo que os caracteres não ASCII sejam utilizados dentro de comentários ou cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

- Não é permitida a utilização de qualquer módulo ou função não disponível built-in no Python 3.

- 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 (projectos incluídos) leva à reprovação na disciplina. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projecto.

4 Submissão

A submissão e avaliação da execução do projecto de FP é feita utilizando o sistema Mooshak [3]. Para obter as necessárias credenciais de acesso e poder usar o sistema deverá:

- Obter uma password para acesso ao sistema, seguindo as instruções na página: `http://acm.tecnico.ulisboa.pt/~fpshak/cgi-bin/getpass`. A password ser-lhe-á enviada para o email que tem configurado no Fenix. Se a password 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 projecto. 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 projecto podendo ver o número de testes passados/falhados.
- Para sair do sistema utilize o botão *"Logout"*.

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

- 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 [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 projecto que pretende que seja avaliada. Não há excepções!

[3] A versão de Python utilizada nos testes automáticos é Python 3.5.3.

[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.

- Cada aluno tem direito a **10 submissões sem penalização** no Mooshak. Por cada submissão adicional serão descontados 0.1 valores na componente de avaliação automática.

5 Classificação

A nota do projecto será baseada nos seguintes aspetos:

1. **Avaliação automática (80%).** A avaliação da correcta execução será feita através do sistema Mooshak. O tempo de execução de cada teste está limitado, bem como a memória utilizada.
   Existem 162 casos de teste configurados no sistema: 33 testes públicos (disponibilizados na página da disciplina) valendo 0 pontos cada e 129 testes privados (não disponibilizados). Como a avaliação automática vale 80% (equivalente a 16 valores) da nota, uma submissão obtém a nota máxima de 1600 pontos.
   O facto de um projecto completar com sucesso os testes públicos fornecidos não implica que esse projecto 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. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura[5]. Em particular, serão consideradas as seguintes componentes:

   - Boas práticas (1.5 valores): serão considerados entre outros a clareza do código, a integração de conhecimento adquirido durante a UC e a criatividade das soluções propostas.
   - Comentários (1 valor): deverão incluir a assinatura das funções definidas, 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 projecto):

- 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.

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

- No processo de desenvolvimento do projecto, 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 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);

- 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.