TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2023-24
Segundo Projeto
20 de Outubro de 2023

Go

O segundo projeto de Fundamentos da Programação consiste em escrever um programa em Python que permita jogar ao Go[1]. Para este efeito, deverá definir um conjunto de tipos abstratos de dados que deverão ser utilizados para manipular a informação necessária no decorrer do jogo, bem como um conjunto de funções adicionais.

1 Descrição do jogo

O Go é um jogo de tabuleiro de estratégia abstrato para dois jogadores. Os jogadores colocam alternadamente pedras da sua cor no tabuleiro, com o objetivo básico do formar territórios ao redor de regiões vazias do tabuleiro. Ganha quem atingir a maior pontuação, o que basicamente corresponde ao controle de um território maior. O jogo foi inventado na China há mais de 2500 anos e acredita-se que seja o jogo de tabuleiro mais antigo ainda jogado nos tempos modernos.

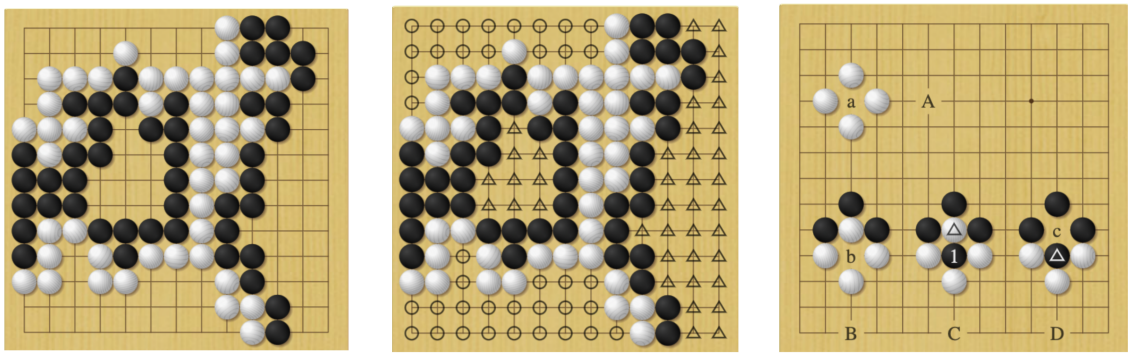
1.1 Goban, interseções e pedras

O tabuleiro de Go ou **goban** é uma estrutura retangular de 19×19 linhas que formam **interseções**, embora também seja comum usar tabuleiros menores, de tamanho 13×13, e até 9×9. As interseções são identificadas por uma letra maiúscula de A até S, e por um número do 1 até 19. Duas interseções são ditas **adjacentes** se forem conectadas por uma linha horizontal ou vertical sem outras interseções entre elas. Uma interseção pode estar livre ou ocupada pela **pedra** de um dos jogadores (branca ou preta, dependendo do jogador). A Figura 1 mostra vários exemplos de tabuleiros de Go. A **ordem de leitura** das interseções do goban é sempre feita da esquerda para a direita seguida de baixo para cima.

1.2 Cadeias, liberdades e territórios

Duas interseções com pedras brancas, pretas ou *livres* estão **conetadas** se for possível traçar um percurso desde uma interseção para a outra passando sempre por interseções adjacentes ocupadas por pedras brancas, ocupadas por pedras pretas ou *livres*, respetivamente. Uma **cadeia de pedras** é um conjunto de uma ou mais interseções ocupadas por pedras (necessariamente da mesma cor) que estão todas conetadas entre si e que não estão conetadas a nenhuma outra pedra da mesma cor. As **liberdades** de uma pedra

[1] `https://en.wikipedia.org/wiki/Go_(game)`



a) b) c)

Figura 1: a) Tabuleiro de go de tamanho 13×13. b) Tabuleiro de go com as interseções dos territórios do jogador branco assinaladas com círculos, e as do jogador preto com triângulos. c) Exemplos de jogadas: (A) O jogador preto não pode jogar em a (regra do suicídio), mas poderia jogar em b. As imagens (B), (C) e (D) mostram a sequência de colocação e captura após o jogador preto jogar em b. O jogador branco, não poderia jogar imediatamente em c porque resultaria na repetição do anterior estado do tabuleiro (regra da repetição ou *ko*).

é o conjunto de interseções livres adjacentes a essa pedra ou adjacente a uma pedra da mesma cadeia. Analogamente às cadeias de pedras, definimos um **território** como o conjunto maximal de uma ou mais interseções livres que estão todas conetadas entre si e que não estão conetadas a nenhuma outra interseção livre. Chamamos **fronteira** de um território ao conjunto de todas às interseções ocupadas por pedras adjacentes a um território. Diz-se que um território pertence a um jogador se a sua fronteira estiver ocupada apenas por pedras da cor desse jogador.

1.3 Regras do jogo

Para a realização deste projeto, consideraremos as seguintes regras do jogo Go:

1. **Início:** No início do jogo, o tabuleiro está vazio. O jogador com pedras pretas é o primeiro a jogar. A seguir, os jogadores alternam em turnos subsequentes.

2. **Turno:** No seu turno, um jogador pode passar a vez ou jogar. Uma jogada de um jogador consiste nas seguintes etapas (realizadas em ordem):

   (a) *Colocar:* Coloca uma pedra da sua cor numa intersecção vazia. As pedras não podem ser movidas para outra interseção após serem jogadas.

   (b) *Capturar:* Retira do tabuleiro quaisquer pedras da cor do oponente que não tenham liberdades.

3. **Jogadas ilegais:** As seguintes restrições devem ser consideradas na colocação das pedras:

(a) *Suicídio:* Uma jogada de um jogador é ilegal se uma ou mais pedras da cor daquele jogador ficarem sem liberdades após a resolução da jogada.

(b) *Repetição (ko):* Uma jogada é ilegal se tiver o efeito (após todas as etapas da jogada terem sido concluídas) de criar um estado do tabuleiro que ocorreu anteriormente no jogo.

4. **Fim e pontuação:** O jogo termina quando ambos os jogadores tiverem passado a vez consecutivamente. No estado final do tabuleiro, a pontuação de um jogador é obtida como a soma do número total de interseções que:

   (a) Pertencem ao território desse jogador;

   (b) Estão ocupadas por uma pedra da cor daquele jogador.

5. **Vencedor** O jogador com maior pontuação é o vencedor. Se os dois jogadores obtiverem a mesma pontuação, o jogador branco ganha.

2 Trabalho a realizar

Um dos objetivos deste segundo projeto é definir e implementar 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 jogo Go.

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.
- As barreiras de abstração devem ser sempre respeitadas.

2.1.1 TAD *intersecao* (1,5 valores)

O TAD imutável *intersecao* é usado para representar uma intersecção do tabuleiro de Go. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_intersecao*: $str \times int \mapsto intersecao$
    cria_intersecao(col,lin) recebe um caracter e um inteiro correspondentes à coluna *col* e à linha *lin* e devolve a interseção correspondente. O construtor verifica a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_intersecao: argumentos invalidos'` caso os seus argumentos não sejam válidos.

- Seletores
  - *obtem_col*: *intersecao* $\mapsto$ *str*
    obtem_col(i) devolve a coluna *col* da interseção i.
  - *obtem_lin*: *intersecao* $\mapsto$ *int*
    obtem_lin(i) devolve a linha *lin* da interseção i.
- Reconhecedor
  - *eh_intersecao*: *universal* $\mapsto$ *booleano*
    eh_intersecao(arg) devolve `True` caso o seu argumento seja um TAD *intersecao* e `False` caso contrário.
- Teste
  - *intersecoes_iguais*: *universal* $\times$*universal* $\mapsto$ *booleano*
    intersecoes_iguais(i1, i2) devolve `True` apenas se $i1$ e $i2$ são interseções e são iguais, e `False` caso contrário.
- Transformador
  - *intersecao_para_str*: *intersecao* $\mapsto$ *str*
    intersecao_para_str(i) devolve a cadeia de caracteres que representa o seu argumento, como mostrado nos exemplos.
  - *str_para_intersecao*: *str* $\mapsto$ *intersecao*
    str_para_intersecao(s) devolve a interseção representada pelo seu argumento.

As funções de alto nível associadas a este TAD são:

- *obtem_intersecoes_adjacentes*: *intersecao* $\times$ *intersecao* $\mapsto$ *tuplo*
  obtem_intersecoes_adjacentes(i, l) devolve um *tuplo* com as interseções adjacentes à interseção i de acordo com a ordem de leitura em que l corresponde à interseção superior direita do tabuleiro de Go.
- *ordena_intersecoes*: *tuplo* $\mapsto$ *tuplo*
  ordena_intersecoes(t) devolve um *tuplo* de interseções com as mesmas interseções de t ordenadas de acordo com a ordem de leitura do tabuleiro de Go.

Exemplos de interação:

```
>>> i1 = cria_intersecao('a', 12)
Traceback (most recent call last): <...>
ValueError: cria_intersecao: argumentos invalidos
>>> i1 = cria_intersecao('A', 2)
>>> i2 = cria_intersecao('B', 13)
```

```
>>> intersecoes_iguais(i1, i2)
False
>>> intersecao_para_str(i2)
'B13'
>>> intersecoes_iguais(i1, str_para_intersecao('A2'))
True
>>> tuple(intersecao_para_str(i) \
    for i in obtem_intersecoes_adjacentes(i1, cria_intersecao('S',19))
('A1', 'B2', 'A3')
>>> tup = (cria_intersecao('A',1), cria_intersecao('A',3),
           cria_intersecao('B',1), cria_intersecao('B',2))
>>> tuple(intersecao_para_str(i) for i in ordena_intersecoes(tup))
('A1', 'B1', 'B2', 'A3')
```

2.1.2 TAD *pedra* (1,5 valores)

O TAD *pedra* é usado para representar as pedras do Go. As pedras podem pertencer ao jogador branco ('O') ou ao jogador preto ('X'). Por conveniência, é também definido o conceito pedra *neutra*, que é uma pedra que não pertence a nenhum jogador. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_pedra_branca*: {} ↦ *pedra*
    cria_pedra_branca() devolve uma pedra pertencente ao jogador branco.
  - *cria_pedra_preta*: {} ↦ *pedra*
    cria_pedra_preta() devolve uma pedra pertencente ao jogador preto.
  - *cria_pedra_neutra*: {} ↦ *pedra*
    cria_pedra_neutra() devolve uma pedra neutra.
- Reconhecedor
  - *eh_pedra*: *universal* ↦ *booleano*
    eh_pedra(arg) devolve `True` caso o seu argumento seja um TAD *pedra* e `False` caso contrário.
  - *eh_pedra_branca*: *pedra* ↦ *booleano*
    eh_pedra_branca(p) devolve `True` caso a *pedra p* seja do jogador branco e `False` caso contrário.
  - *eh_pedra_preta*: *pedra* ↦ *booleano*
    eh_pedra_preta(p) devolve `True` caso a *pedra p* seja do jogador preto e `False` caso contrário.
- Teste

  - *pedras_iguais*: *universal* $\times$*universal* $\mapsto$ *booleano*
    pedras_iguais(p1, p2) devolve `True` apenas se $p1$ e $p2$ são pedras e são iguais.

- Transformador
  - *pedra_para_str*: *pedra* $\mapsto$ *str*
    pedra_para_str(p) devolve a cadeia de caracteres que representa o jogador dono da pedra, isto é, 'O', 'X' ou '.' para pedras do jogador branco, preto ou neutra respetivamente.

As funções de alto nível associadas a este TAD são:

- *eh_pedra_jogador*: *pedra* $\mapsto$ *booleano*
  eh_pedra_jogador(p) devolve `True` caso a pedra p seja de um jogador e `False` caso contrário.

Exemplos de interação:

```
>>> b = cria_pedra_branca()
>>> eh_pedra(b)
True
>>> p = cria_pedra_preta()
>>> pedras_iguais(b, p)
False
>>> pedra_para_str(b), pedra_para_str(p)
('O', 'X')
>>> eh_pedra_jogador(cria_pedra_neutra())
False
```

2.1.3 TAD *goban* (5,0 valores)

O TAD *goban* é usado para representar um tabuleiro do jogo Go e as pedras dos jogadores que nele são colocadas. As operações básicas associadas a este TAD são:

- Construtor
  - *cria_goban_vazio*: *int* $\mapsto$ *goban*
    cria_goban_vazio(n) devolve um *goban* de tamanho $n \times n$, sem interseções ocupadas. O construtor verifica a validade do argumento, gerando um `ValueError` com a mensagem `'cria_goban_vazio: argumento invalido'` caso os seu argumento não seja válido. Considere que um *goban* pode ser de dimensão 9×9, 13×13 ou 19×19.
  - *cria_goban*: *int* $\times$*tuplo* $\times$*tuplo* $\mapsto$ *goban*
    cria_goban(n, ib, ip) devolve um *goban* de tamanho $n \times n$, com as interseções do tuplo *ib* ocupadas por pedras brancas e as interseções do tuplo *ip* ocupadas

por pedras pretas. O construtor verifica a validade dos argumentos, gerando um `ValueError` com a mensagem `'cria_goban: argumentos invalidos'` caso os seus argumentos não sejam válidos. Considere que um *goban* pode ser de dimensão 9×9, 13×13 ou 19×19.

- *cria_copia_goban*: *goban* $\mapsto$ *goban*
  cria_copia_goban(t) recebe um *goban* e devolve uma cópia do *goban*.

- Seletores
  - *obtem_ultima_intersecao*: *goban* $\mapsto$ *intersecao*
    obtem_ultima_intersecao(g) devolve a interseção que corresponde ao canto superior direito do goban g.
  - *obtem_pedra*: *goban* $\times$ *intersecao* $\mapsto$ *pedra*
    obtem_pedra(g, i) devolve a pedra na interseção i do goban g. Se a interseção não estiver ocupada, devolve uma pedra *neutra*.
  - *obtem_cadeia*: *goban* $\times$ *intersecao* $\mapsto$ *tuplo*
    obtem_cadeia(g, i) devolve o tuplo formado pelas interseções (em ordem de leitura) das pedras da mesma cor que formam a cadeia que passa pela interseção i. Se a posição não estiver ocupada, devolve a cadeia de interseções livres.

- Modificadores
  - *coloca_pedra*: *goban* $\times$ *intersecao* $\times$*pedra* $\mapsto$ *goban*
    coloca_pedra(g, i, p) modifica destrutivamente o goban g colocando a pedra do jogador p na interseção i, e devolve o próprio *goban*.
  - *remove_pedra*: *goban* $\times$ *intersecao* $\mapsto$ *goban*
    remove_pedra(g, i) modifica destrutivamente o goban g removendo a pedra da interseção i, e devolve o próprio *goban*.
  - *remove_cadeia*: *goban* $\times$ *tuplo* $\mapsto$ *goban*
    remove_cadeia(g, t) modifica destrutivamente o goban g removendo as pedras nas interseções to tuplo t, e devolve o próprio *goban*.

- Reconhecedor
  - *eh_goban*: *universal* $\mapsto$ *booleano*
    eh_goban(arg) devolve `True` caso o seu argumento seja um TAD *goban* e `False` caso contrário.
  - *eh_intersecao_valida*: *goban* $\times$*intersecao* $\mapsto$ *booleano*
    eh_intersecao_valida(g, i) devolve `True` se i é uma interseção válida dentro do goban g e `False` caso contrário.

- Teste

  - *gobans_iguais*: *universal* $\times$ *universal* $\mapsto$ *booleano*
    gobans_iguais(g1, g2) devolve `True` apenas se $g1$ e $g2$ forem *gobans* e forem iguais.

- Transformador
  - *goban_para_str*: *goban* $\mapsto$ *str*
    goban_para_str(g) devolve a cadeia de caracteres que representa o *goban* como mostrado nos exemplos.

As funções de alto nível associadas a este TAD são:

- *obtem_territorios*: *goban* $\mapsto$ *tuplo*
  obtem_territorios(g) devolve o tuplo formado pelos tuplos com as interseções de cada território de g. A função devolve as interseções de cada território ordenadas em ordem de leitura do tabuleiro de Go, e os territórios ordenados em ordem de leitura da primeira interseção do território.

- *obtem_adjacentes_diferentes*: *goban* $\times$ *tuplo* $\mapsto$ *tuplo*
  obtem_adjacentes_diferentes(g, t) devolve o tuplo ordenado formado pelas interseções adjacentes às interseções do tuplo t:
  (a) livres, se as interseções do tuplo t estão ocupadas por pedras de jogador;
  (b) ocupadas por pedras de jogador, se as interseções do tuplo t estão livres.
  Notar que o primeiro caso corresponde às *liberdades* de uma cadeia de pedras, enquanto que o segundo corresponde à *fronteira* de um território.

- *jogada*: *goban* $\times$*intersecao* $\times$*pedra* $\mapsto$ *goban*
  jogada(g, i, p) modifica destrutivamente o goban g colocando a pedra de jogador p na interseção i e remove todas as pedras do jogador contrário pertencentes a cadeias adjacentes à i sem liberdades, devolvendo o próprio *goban*.

- *obtem_pedras_jogadores*: *goban* $\mapsto$ *tuplo*
  obtem_pedras_jogadores(g) devolve um *tuplo* de dois inteiros que correspondem ao número de interseções ocupadas por pedras do jogador branco e preto, respetivamente.

Exemplos de interação:

```
>>> g = cria_goban_vazio(10)
Traceback (most recent call last): <...>
ValueError: cria_goban_vazio: argumento invalido
>>> g = cria_goban_vazio(9)
>>> i1 = cria_intersecao('C',8)
```

```
>>> pedra_para_str(obtem_pedra(g,i1))
'.'
>>> b, p = cria_pedra_branca(), cria_pedra_preta()
>>> ib = 'C1', 'C2', 'C3', 'D2', 'D3', 'D4', 'A3', 'B3'
>>> ip = 'E4', 'E5', 'F4', 'F5', 'G6', 'G7'
>>> for i in ib: coloca_pedra(g, str_para_intersecao(i), b)
>>> for i in ip: coloca_pedra(g, str_para_intersecao(i), p)
>>> print(goban_para_str(g))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
 6 . . . . . . X . .  6
 5 . . . . X X . . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 . . O O . . . . .  2
 1 . . O . . . . . .  1
   A B C D E F G H I
>>> cad = obtem_cadeia(g, cria_intersecao('F',5))
>>> tuple(intersecao_para_str(i) for i in cad)
('E4', 'F4', 'E5', 'F5')
>>> liberdades = obtem_adjacentes_diferentes(g, cad)
>>> tuple(intersecao_para_str(i) for i in liberdades)
('E3', 'F3', 'G4', 'D5', 'G5', 'E6', 'F6')
>>> terr = obtem_territorios(g)
>>> tuple(intersecao_para_str(i) for i in terr[0])
('A1', 'B1', 'A2', 'B2')
>>> border = obtem_adjacentes_diferentes(g, terr[0])
>>> tuple(intersecao_para_str(i) for i in border)
('C1', 'C2', 'A3', 'B3')
>>> obtem_pedras_jogadores(g)
(8, 6)
>>> ib = tuple(str_para_intersecao(i) \
    for i in ('C1', 'C2', 'C3', 'D2', 'D3', 'D4', 'A3', 'B3'))
>>> ip = tuple(str_para_intersecao(i) \
    for i in ('A1', 'A2', 'B1', 'E4', 'E5', 'F4', 'F5', 'G6', 'G7'))
>>> g = cria_goban(9, ib, ip)
>>> print(goban_para_str(g))
  A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
```

```
 6 . . . . . . X . .  6
 5 . . . . X X . . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 X . O O . . . . .  2
 1 X X O . . . . . .  1
   A B C D E F G H I
>>> g2 = jogada(g, cria_intersecao('B', 2), b)
>>> print(goban_para_str(g))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
 6 . . . . . . X . .  6
 5 . . . . X X . . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 . O O O . . . . .  2
 1 . . O . . . . . .  1
   A B C D E F G H I
```

2.2 Funções adicionais

2.2.1 calcula_pontos: *goban* $\mapsto$ *tuple* (0,5 valores)

calcula_pontos(g) é uma função auxiliar que recebe um *goban* e devolve o tuplo de dois inteiros com as pontuações dos jogadores branco e preto, respetivamente.

```
>>> ib = tuple(str_para_intersecao(i) \
    for i in ('C1', 'C2', 'C3', 'D2', 'D3', 'D4', 'A3', 'B3'))
>>> ip = tuple(str_para_intersecao(i) \
    for i in ('E4', 'E5', 'F4', 'F5', 'G6', 'G7'))
>>> g = cria_goban(9, ib, ip)
>>> calcula_pontos(g)
(12, 6)
```

2.2.2 eh_jogada_legal: *goban* $\times$*intersecao* $\times$ *pedra*$\times$ *goban* $\mapsto$ *booleano* (1,0 valores)

eh_jogada_legal(g, i, p, l) é uma função auxiliar que recebe um *goban* g, uma interseção i, uma pedra de jogador p e um outro *goban* l e devolve `True` se a jogada for legal ou `False` caso contrário, sem modificar g ou l. Para a deteção de repetição, considere que l

representa o estado de tabuleiro que não pode ser obtido após a resolução completa da jogada.

```
>>> ib = tuple(str_para_intersecao(i) \
    for i in ('C1', 'C2', 'C3', 'D2', 'D3', 'D4', 'A3', 'B3'))
>>> ip = tuple(str_para_intersecao(i) \
    for i in ('A1', 'A2', 'B1', 'E4', 'E5', 'F4', 'F5', 'G6', 'G7'))
>>> g = cria_goban(9, ib, ip)
>>> l = cria_goban_vazio(9)
>>> b, p = cria_pedra_branca(), cria_pedra_preta()
>>> eh_jogada_legal(g, cria_intersecao('B', 2), p, l)
False
>>> eh_jogada_legal(g, cria_intersecao('B', 2), b, l)
True
>>> print(goban_para_str(g))
  A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
 6 . . . . . . X . .  6
 5 . . . . X X . . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 X . O O . . . . .  2
 1 X X O . . . . . .  1
   A B C D E F G H I
>>> ib = tuple(str_para_intersecao(i) for i in ('A2','B1','B3','C2'))
>>> ip = tuple(str_para_intersecao(i) for i in ('C1','C3','D1','D2'))
>>> g = cria_goban(9, ib, ip)
>>> print(goban_para_str(g))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . O X . . . . . .  3
 2 O . O X . . . . .  2
 1 . O X X . . . . .  1
   A B C D E F G H I
>>> g_ko = cria_copia_goban(g)
>>> eh_jogada_legal(g, cria_intersecao('B', 2), p, g_ko)
True
```

```
>>> print(goban_para_str(jogada(g, cria_intersecao('B', 2), p)))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . O X . . . . . .  3
 2 O X . X . . . . .  2
 1 . O X X . . . . .  1
   A B C D E F G H I
>>> eh_jogada_legal(g, cria_intersecao('B', 2), b, g_ko)
False
```

2.2.3 turno_jogador: *goban* × *pedra* × *goban* ↦ *booleano* (1,0 valores)

turno_jogador(g, p, l) é uma função auxiliar que recebe um *goban* g, uma pedra de jogador p e um outro *goban* l, e oferece ao jogador que joga com pedras p a opção de passar ou de colocar uma pedra própria numa interseção. Se o jogador passar, a função devolve `False` sem modificar os argumentos. Caso contrário, a função devolve `True` e modifica destrutivamente o tabuleiro g de acordo com a jogada realizada. A função deve apresentar a mensagem do exemplo a seguir, repetindo a mensagem até que o jogador introduzir `'P'` ou a representação externa de uma interseção do tabuleiro de Go que corresponda a uma jogada legal. Considere que l representa o estado de tabuleiro que não pode ser obtido após a resolução completa da jogada.

```
>>> ib = tuple(str_para_intersecao(i)
   for i in ('C1', 'C2', 'C3', 'D2', 'D3', 'D4', 'A3', 'B3'))
>>> ip = tuple(str_para_intersecao(i)
   for i in ('A1', 'A2', 'B1', 'E4', 'E5', 'F4', 'F5', 'G6', 'G7'))
>>> g = cria_goban(9, ib, ip)
>>> print(goban_para_str(g))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
 6 . . . . . . X . .  6
 5 . . . . X X . . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 X . O O . . . . .  2
 1 X X O . . . . . .  1
```

```
   A B C D E F G H I
>>> turno_jogador(g, cria_pedra_preta(),cria_goban_vazio(9))
Escreva uma intersecao ou 'P' para passar [X]:B10
Escreva uma intersecao ou 'P' para passar [X]:B2
Escreva uma intersecao ou 'P' para passar [X]:G5
True
>>> print(goban_para_str(g))
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . X . .  7
 6 . . . . . . X . .  6
 5 . . . . X X X . .  5
 4 . . . O X X . . .  4
 3 O O O O . . . . .  3
 2 X . O O . . . . .  2
 1 X X O . . . . . .  1
   A B C D E F G H I
```

2.2.4 go: *int* × *tuple* × *tuple* ↦ *booleano* (1,5 valores)

go(n, tb, tn) é a função principal que permite jogar um jogo completo do Go de dois jogadores. A função recebe um inteiro correspondente à dimensão do tabuleiro, e dois tuplos (potencialmente vazios) com a representação externa das interseções ocupadas por pedras brancas (*tb*) e pretas (*tp*) inicialmente. O jogo termina quando os dois jogadores passam a vez de jogar consecutivamente. A função devolve `True` se o jogador com pedras brancas conseguir ganhar o jogo, ou `False` caso contrário. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'go: argumentos invalidos'` caso os seus argumentos não sejam válidos.

Exemplo 1

```
>>> go(9, (), ())
Branco (O) tem 0 pontos
Preto (X) tem 0 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . . . . . . . . .  3
 2 . . . . . . . . .  2
```

```
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:A1
Branco (O) tem 0 pontos
Preto (X) tem 81 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . . . . . . . . .  3
 2 . . . . . . . . .  2
 1 X . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:B1
Branco (O) tem 1 pontos
Preto (X) tem 1 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . . . . . . . . .  3
 2 . . . . . . . . .  2
 1 X O . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:B2
Branco (O) tem 1 pontos
Preto (X) tem 2 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . . . . . .  5
 4 . . . . . . . . .  4
 3 . . . . . . . . .  3
 2 . X . . . . . . .  2
 1 X O . . . . . . .  1
```

```
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:A2
Branco (O) tem 3 pontos
Preto (X) tem 1 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . . . . . . 5
 4 . . . . . . . . . 4
 3 . . . . . . . . . 3
 2 O X . . . . . . . 2
 1 . O . . . . . . . 1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:A1
Escreva uma intersecao ou 'P' para passar [X]:A3
Branco (O) tem 3 pontos
Preto (X) tem 2 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . . . . . . 5
 4 . . . . . . . . . 4
 3 X . . . . . . . . 3
 2 O X . . . . . . . 2
 1 . O . . . . . . . 1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:A1
Branco (O) tem 3 pontos
Preto (X) tem 2 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . . . . . . 5
 4 . . . . . . . . . 4
 3 X . . . . . . . . 3
 2 O X . . . . . . . 2
 1 O O . . . . . . . 1
```

```
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:C1
Branco (O) tem 0 pontos
Preto (X) tem 81 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . . . . . . 5
 4 . . . . . . . . . 4
 3 X . . . . . . . . 3
 2 . X . . . . . . . 2
 1 . . X . . . . . . 1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:E5
Branco (O) tem 1 pontos
Preto (X) tem 6 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . O . . . . 5
 4 . . . . . . . . . 4
 3 X . . . . . . . . 3
 2 . X . . . . . . . 2
 1 . . X . . . . . . 1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:P
Branco (O) tem 1 pontos
Preto (X) tem 6 pontos
   A B C D E F G H I
 9 . . . . . . . . . 9
 8 . . . . . . . . . 8
 7 . . . . . . . . . 7
 6 . . . . . . . . . 6
 5 . . . . O . . . . 5
 4 . . . . . . . . . 4
 3 X . . . . . . . . 3
 2 . X . . . . . . . 2
 1 . . X . . . . . . 1
   A B C D E F G H I
```

```
Escreva uma intersecao ou 'P' para passar [O]:P
Branco (O) tem 1 pontos
Preto (X) tem 6 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . . . . . . . .  8
 7 . . . . . . . . .  7
 6 . . . . . . . . .  6
 5 . . . . O . . . .  5
 4 . . . . . . . . .  4
 3 X . . . . . . . .  3
 2 . X . . . . . . .  2
 1 . . X . . . . . .  1
   A B C D E F G H I
False
```

Exemplo 2

```
>>> ib = 'C2', 'D2', 'E2', 'F2', 'G2', 'H2', 'B3', 'I3', 'B4', 'D4', \
   'E4', 'F4', 'B5', 'D5', 'G5', 'I5', 'B6', 'D6', 'E6', 'F6', 'G6', \
   'I6', 'C7', 'I7', 'C8', 'D8', 'E8', 'F8', 'G8', 'H8', 'I8'
>>> ip = 'C3', 'D3', 'E3', 'F3', 'G3', 'C4', 'G4', 'H4', 'C5', 'H5', \
   'C6', 'H6', 'D7', 'E7', 'F7', 'G7', 'H7'
>>> go(9, ib, ip)
Branco (O) tem 62 pontos
Preto (X) tem 17 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
 7 . . O X X X X X O  7
 6 . O X O O O O X O  6
 5 . O X O . . O X O  5
 4 . O X O O O X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:E5
Branco (O) tem 60 pontos
Preto (X) tem 18 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
```

```
 7 . . O X X X X X O  7
 6 . O X O O O O X O  6
 5 . O X O X . O X O  5
 4 . O X O O O X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:F5
Branco (O) tem 62 pontos
Preto (X) tem 17 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
 7 . . O X X X X X O  7
 6 . O X O O O O X O  6
 5 . O X O . O O X O  5
 4 . O X O O O X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:E5
Branco (O) tem 51 pontos
Preto (X) tem 28 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
 7 . . O X X X X X O  7
 6 . O X . . . . X O  6
 5 . O X . X . . X O  5
 4 . O X . . . X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [O]:P
Branco (O) tem 51 pontos
Preto (X) tem 28 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
 7 . . O X X X X X O  7
```

```
 6 . O X . . . . X O  6
 5 . O X . X . . X O  5
 4 . O X . . . X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
Escreva uma intersecao ou 'P' para passar [X]:P
Branco (O) tem 51 pontos
Preto (X) tem 28 pontos
   A B C D E F G H I
 9 . . . . . . . . .  9
 8 . . O O O O O O O  8
 7 . . O X X X X X O  7
 6 . O X . . . . X O  6
 5 . O X . X . . X O  5
 4 . O X . . . X X .  4
 3 . O X X X X X . O  3
 2 . . O O O O O O .  2
 1 . . . . . . . . .  1
   A B C D E F G H I
True
```

3 Condições de Realização e Prazos

- A entrega do 2º projeto será efetuada exclusivamente por via eletrónica. Para submeter o seu projeto deverá realizar pelo menos uma atualização do repositório remoto GitLab fornecido pelo corpo docente, até às **17:00 do dia 3 de Novembro de 2023**. Depois desta hora, qualquer atualização do repositório será ignorada. Não serão aceites submissões de projetos por outras vias sob pretexto algum.

- A solução do projeto deverá consistir apenas num único ficheiro com extensão *.py* contendo todo o código do seu projeto.

- Cada aluno tem direito a **15 submissões sem penalização**. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.

- 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.

- 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 do projeto de FP é realizada atualizando o repositório remoto GitLab privado fornecido pelo corpo docente para cada aluno (ou grupo de projeto). O endereço web do repositório do projeto dos alunos é `https://gitlab.rnl.tecnico.ulisboa.pt/ist-fp/fp23/prj2/(curso)/(grupo)`, onde:

- `(curso)` pode ser `leic-a`, `leic-t`, `leti` ou `leme`;
- `(grupo)` pode ser:
  - o ist-id para os alunos da LEIC-A, LEIC-T e LETI (ex. `ist190000`);
  - `g` seguido dos dois dígitos que identificam o número de grupo de projeto dos alunos da LEME (ex. `g00`).

Sempre que é realizada uma nova atualização do repositório remoto é desencadeado o processo de avaliação automática do projeto e é contabilizada uma nova submissão. Quando a submissão tiver sido processada, poderá visualizar um relatório de execução com os detalhes da avaliação automática do seu projeto em `http://fp.rnl.tecnico.ulisboa.pt/reports/(grupo)/`. Adicionalmente, receberá no seu email o mesmo relatório. Se não receber o email ou o relatório web aparentar não ter sido atualizado, contacte o corpo docente. Note que o sistema de submissão e avaliação não limita o número de submissões simultâneas. Um número elevado de submissões num determinado momento, poderá ocasionar a rejeição de alguns pedidos de avaliação. Para evitar problemas de último momento, **recomenda-se que submeta o seu projeto atempadamente**.

Detalhes sobre como aceder ao GitLab, configurar o par de chaves SSH, executar os comandos de Git e recomendações sobre ferramentas, encontram-se na página da disciplina na seção "Material de Apoio - Ambiente de Desenvolvimento"[3].

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

[3] `https://fenix.tecnico.ulisboa.pt/disciplinas/FProg3/2023-2024/1-semestre/ambiente-de-desenvolvimento`

5 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. **Execução correta (60%).** A avaliação da correta execução será feita com um conjunto de testes unitários utilizando o módulo de Python pytest[4].
   Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. Como os testes de execução valem 60% (equivalente a 12 valores) da nota do projeto, uma submissão obtém a nota máxima de 1200 pontos por esta componente.
   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 usando testes próprios adicionais, de forma a completar com sucesso os testes privados.

2. **Respeito pelas barreiras de abstração (20%).** A avaliação do respeito pelas barreiras de abstração também será feita automaticamente utilizando o módulo de Python pytest. Para este fim, serão usados um conjunto de testes (diferentes dos testes de execução) especificamente definidos para testar que o código desenvolvido pelos alunos respeita as barreiras de abstração. Como os testes de abstração valem 20% (equivalente a 4 valores) da nota do projeto, uma submissão obtém a nota máxima de 400 pontos por esta componente.

3. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura. Em particular, serão consideradas as seguintes componentes:

   - 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):

[4] `https://docs.pytest.org/en/7.4.x/`

- 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.
- 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 sentir 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.