Computação 1, 2021.1
Trabalho Final - O jogo das 8 rainhas

Para entrega até 18/10 às 10:00

Submeta suas soluções colocando os arquivos correspondentes na sua pasta do Google Drive*

Lembre-se de escolher bons nomes para suas funções e variáveis, de documentar seu código com *docstrings* (documentação de função) e comentários onde for apropriado, e de fazer testes para suas funções (na documentação usando o módulo `doctest` ou em funções dedicadas).

1 O jogo

Neste trabalho, vamos implementar uma variante em forma de jogo do *problema das 8 rainhas*. Este desafio de xadrez foi publicado por Max Bezzel em 1848, e generalizado para n rainhas por Franz Nauck em 1850.[1] [2]

O problema original consiste em encontrar uma disposição de 8 rainhas em um tabuleiro de xadrez 8×8, de forma que nenhuma rainha ataque outra. Para este trabalho, formularemos uma variante (e algumas generalizações) para dois jogadores. O jogo consiste em um tabuleiro $n \times n$, onde dois jogadores devem dispor peças de xadrez alternadamente. No início, cada jogador recebe uma certa quantidade de peças (exemplos adiante). A cada rodada, o jogador escolhe uma peça e uma casa do tabuleiro, desde que, ao colocar a peça nesta casa, ele não seja atacada por nenhuma das peças já presentes, nem ataque nenhuma das peças presentes. Se não for possível para o jogador escolher uma peça e uma casa *válidas*, ou se o jogador colocar uma peça em uma casa *inválida*, ele perde (isso inclui os casos do jogador escolher uma peça que não tenha para jogar, ou de escolher uma posição inexistente no tabuleiro). Se ambos jogadores conseguirem jogar todas as suas peças, vence quem tiver feito a última jogada.

Para facilitar, vamos rotular as linhas do tabuleiro com os números de $N - 1$ a 0 (como em Python), e as colunas também com os números, agora em ordem crescente.

Exemplos de tabuleiros com jogos em andamento podem ser vistos a partir da página 8.

1.1 Os diferentes jogos

Vamos considerar, para este trabalho, as seguintes modalidades do jogo das rainhas:

Tradicional: 4 rainhas para cada jogador, tabuleiro 8 x 8

Grande: 10 rainhas para cada jogador, tabuleiro 20 x 20

*Link recebido por email em 19/7/2021 — o nome é parecido com `<seu nome> - Computação 1 - Submissões e Feedback.`

[1] `https://pt.wikipedia.org/wiki/Problema_das_oito_damas`

[2] `https://en.wikipedia.org/wiki/Eight_queens_puzzle`

Ímpar: 7 rainhas para o primeiro jogador, 6 para o segundo, tabuleiro 13 x 13

X-Random: 5 peças aleatórias para cada jogador, tabuleiro 8 x 8

2 A representação do jogo em Python

Nossos jogos serão jogados em tabuleiros de tamanho $n \times n$, com $n \leq 20$.

Cada *casa* do tabuleiro será representada por uma `tupla` de 2 `int`s, onde o primeiro elemento denota o número da linha e segundo denota a coluna. Portanto, o ponto no canto inferior esquerdo da malha é `(0,0)`, acima deste está o ponto `(1,0)`, ao lado deste último está o ponto `(1,1)`, etc.

Cada *peça* será representada por uma string, de acordo com a notação usual do xadrez em português: `'R'` para o rei, `'D'` para a rainha ("dama"), `'T'` para a torre, `'B'` para o bispo, `'C'` para o cavalo.

Cada *jogador* será denotado pela `string` com seu nome, e por uma lista com as peças que ainda pode jogar.

Cada *jogada* será denotada pela tupla contendo a *peça* e a *casa* onde esta será jogada.

Com esta notação, a jogada feita na primeira rodada do jogo apresentado na Figura 1 (página 8) foi `('D', (3,1))`

O *estado* do jogo será representado por um dicionário de formato

```
jogo = { 'tamanho': <tamanho_do_tabuleiro>,
         'ocupadas': <dicionário_posições_peças>,
         'primeiro': <lista_peças_primeiro>,
          'segundo': <lista_peças_segundo>,
            'nomes': {'primeiro': <nome_do_primeiro>,
                      'segundo': <nome_do_segundo>}}
```

onde:

- `<tamanho_do_tabuleiro>` é um `int` indicando quantas casas há em um lado do tabuleiro;
- `<dicionário_posições_peças>` é um dicionário cujas chaves são as posições ocupadas até aquele momento no jogo, com respectivos conteúdos sendo as peças que ocupam aquelas posições;
- `<lista_peças_primeiro>` é uma lista contendo as peças de que o primeiro jogador ainda dispõe para jogar;
- `<lista_peças_segundo>` é uma lista contendo as peças de que o segundo jogador ainda dispõe para jogar;
- `<nome_do_primeiro>` é a `str`ing correspondente ao nome do primeiro jogador;
- `<nome_do_segundo>` é a `str`ing correspondente ao nome do segundo jogador.

Assim, o jogo apresentado na Figura 3 ao final do documento tem como dicionário

```
{ 'tamanho': 20,
 'ocupadas': {  (4,9): 'D',
                (8,7): 'D',
              (10,10): 'D'},
 'primeiro': ['D', 'D', 'D', 'D', 'D', 'D', 'D', 'D'],
  'segundo': ['D', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'D'],
    'nomes': {'primeiro': 'Batman',
              'segundo': 'Super-Homem'}}
```

3 O trabalho

Baixe o arquivo

`https://www.hugonobrega.com/seu_nome_aqui.py`

e o renomeie com o seu nome (mantendo a terminação `.py`). Este arquivo contém as seguintes funções prontas para uso:

- `mostra_jogo(jogo)`

  que recebe como entrada um `jogo` e o apresenta na tela, como visto na página 8.

- `outro(jogador)`

  que recebe como entrada uma `string` `'primeiro'` ou `'segundo'` e retorna a "outra" string, i.e., `'segundo'` ou `'primeiro'`, respectivamente.

- `roda_jogo(dict_jogadores=None, modalidade='tradicional', interativo=True)`

  que recebe três entradas:

  - `dict_jogadores`, um dicionário com duas chaves `string`, indicando (em ordem) os dois nomes dos jogadores, e cujos respectivos conteúdos são as funções que indicarão as jogadas de cada jogador (mais detalhes abaixo). Caso esse argumento não seja fornecido, o jogo será entre o jogador humano (Questão 7) chamado `'primeiro'` e o jogador humano chamado `'segundo'`;
  - `modalidade`, uma `str`ing: o tipo de jogo a ser jogado (mais detalhes abaixo);
  - `interativo`, um `bool`: caso `True`, a cada rodada a situação do jogo é mostrada na tela, e ao final o resultado é impresso na tela; caso `False`, o jogo corre sem nada ser impresso na tela. Em ambos os casos o nome do vencedor é retornado.

  A função `roda_jogo` faz, então, a execução do jogo em si, recebendo as jogadas dos jogadores, validando-as, registrando-as, e controlando se o jogo deve continuar ou já acabou (mais detalhes abaixo).

- `torneio(dict_jogadores, partidas=None)`

  que recebe duas entradas:

  - `dict_jogadores`, um dicionário como na função `roda_jogo`, mas com uma quantidade arbitrária de jogadores (desde que pelo menos 2)
  - `partidas`, uma lista de tuplas, onde a primeira entrada é a `str`ing com a modalidade de jogo, e a segunda um `int` com o número de vezes que será repetido, com cada jogador jogando como primeiro e como segundo;

  A função `torneio` faz, então, para cada tipo de jogo em `partidas`, a quantidade indicada de jogos entre cada par de jogadores, e ao final retorna um dicionário indicando as porcentagens de vitórias obtidas.

Para funcionar, a função `roda_jogo` precisa que diversas funções auxiliares sejam implementadas (as funções cujos corpos são apenas a palavra `pass`, no arquivo `seu_nome_aqui.py`), e esta é a primeira parte do seu trabalho (detalhes nas questões abaixo).

Em cada caso, lembre-se de fazer documentação da sua função, de adicionar comentários onde considerar relevante e de fazer funções de teste, quando apropriado.

3.1 Funções auxiliares

Implemente as seguintes funções, necessárias para a execução de jogos e torneios, mas que também serão úteis na próxima seção. É importante que você não altere os nomes das funções nem as ordens de seus argumentos (mas, se você quiser, pode mudar os nomes dos argumentos).

Questão 1. `pos_atacadas_por(tipo_peça, pos_peça, tam_tab=8)`
Recebe

- um tipo de peça (uma string de comprimento 1)
- uma posição (uma tupla de 2 ints)
- o tamanho do tabuleiro (um int, com valor padrão 8)

e retorna um conjunto contendo exatamente as posições do tabuleiro daquele tamanho que seriam atacadas por uma peça daquele tipo a partir daquela posição (portanto, sem incluir a própria posição informada!).

Exemplos:

```
>>> pos_atacadas_por('D', (2,3), 8)
{(0, 1), (2, 4), (1, 2), (2, 1), (2, 7), (4, 3), (3, 4), (7, 3), (6, 7),
(0, 5), (2, 2), (3, 2), (2, 5), (1, 3), (4, 1), (0, 3), (2, 0), (1, 4),
(4, 5), (3, 3), (2, 6), (5, 0), (5, 6), (5, 3), (6, 3)}

>>> pos_atacadas_por('C', (0,5), 8)
{(1, 7), (1, 3), (2, 4), (2, 6)}
```

Questão 2. `todas_pos_atacadas(jogo)`
Recebe

- um jogo (um dicionário, como explicado acima)

e retorna um conjunto contendo exatamente as posições do tabuleiro daquele jogo que estão sendo atacadas por suas peças.

Exemplo:

```
>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'D', 'D'], 'segundo': ['D', 'D', 'D'],
'ocupadas': {(4, 4): 'D', (3, 0): 'D'}, 'nomes': {'primeiro': 'Hugo',
'segundo': 'Bernardo'}}
>>> todas_pos_atacadas(jogo)
{(0, 0), (0, 3), (0, 4), (1, 0), (1, 1), (1, 2), (1, 4), (1, 7), (2, 0),
(2, 1), (2, 2), (2, 4), (2, 6), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5),
(3, 6), (3, 7), (4, 0), (4, 1), (4, 2), (4, 3), (4, 5), (4, 6), (4, 7),
(5, 0), (5, 2), (5, 3), (5, 4), (5, 5), (6, 0), (6, 2), (6, 3), (6, 4),
(6, 6), (7, 0), (7, 1), (7, 4), (7, 7)}
```

Questão 3. `pos_válidas_para(tipo_peça, jogo)`
Recebe

- um tipo de peça (uma string de comprimento 1)
- um jogo (um dicionário)

e retorna um conjunto contendo exatamente as posições do tabuleiro daquele jogo onde seria válido jogar uma peça daquele tipo.

Exemplo:

```
>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'R', 'R', 'T'], 'segundo': ['D', 'D', 'R', 'T'],
'ocupadas': {(3, 0): 'D', (6, 6): 'R'}, 'nomes': {'primeiro': 'Hugo',
'segundo': 'Bernardo'}}
>>> pos_válidas_para('T', jogo)
{(0, 1), (0, 2), (0, 4), (0, 5), (0, 7), (1, 1), (1, 3), (1, 4), (1, 5), (1, 7),
(2, 2), (2, 3), (2, 4), (2, 5), (2, 7), (4, 2), (4, 3), (4, 4), (4, 5), (4, 7),
(5, 1), (5, 3), (5, 4), (7, 1), (7, 2), (7, 3)}
```

Questão 4. `faz_jogada(jogo, jogador, peça, posição)`

Recebe

- um jogo (um dicionário)
- uma string `'primeiro'` ou `'segundo'` (indicando de quem é a vez de jogar)
- uma peça (uma string de comprimento 1)
- uma posição (uma tupla de 2 ints)

e *modifica* o jogo, fazendo nele as alterações necessárias para registrar que aquela jogada foi feita pelo jogador daquela vez. Atenção: você pode assumir que a posição indicada ainda não contém nenhuma peça, que a posição indicada faz sentido no tabuleiro daquele tamanho e que o jogador possui a peça indicada para fazer a jogada. Além disso, você não deve verificar se a jogada é válida.

Exemplo:

```
>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'R', 'R', 'T'], 'segundo': ['D', 'D', 'R', 'T'],
'ocupadas': {(3, 0): 'D', (6, 6): 'R'}, 'nomes': {'primeiro': 'Hugo',
'segundo': 'Bernardo'}}
>>> faz_jogada(jogo, 'primeiro', 'T', (1,7))
>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'R', 'R'], 'segundo': ['D', 'D', 'R', 'T'],
'ocupadas': {(3, 0): 'D', (6, 6): 'R', (1, 7): 'T'}, 'nomes': {'primeiro': 'Hugo',
'segundo': 'Bernardo'}}
```

Questão 5. `jogo_acabou(jogo, último_a_jogar)`

Recebe

- um jogo (um dicionário)
- uma string `'primeiro'` ou `'segundo'` (indicando quem foi o último a jogar)

e retorna:

1. False, caso o jogo ainda nao tenha terminado
2. o vencedor (i.e., a string `'primeiro'` ou `'segundo'`), caso o jogo tenha sido vencido por alguém (note: dependendo da situação, a última jogada pode ter sido vencedora ou perdedora).

Exemplos:

```
>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'R', 'R'], 'segundo': ['D', 'D', 'R', 'T'],
'ocupadas': {(3, 0): 'D', (6, 6): 'R', (1, 7): 'T'}, 'nomes': {'primeiro': 'Hugo',
'segundo': 'Bernardo'}}
>>> jogo_acabou(jogo, 'primeiro')
False

>>> jogo
{'tamanho': 8, 'primeiro': ['D', 'R', 'R'], 'segundo': ['D', 'R', 'T'],
'ocupadas': {(3, 0): 'D', (6, 6): 'R', (1, 7): 'T', (7, 1): 'D'},
'nomes': {'primeiro': 'Hugo', 'segundo': 'Bernardo'}}
>>> jogo_acabou(jogo, 'segundo')
'primeiro'
```

Questão 6. `cria_jogo(modalidade='tradicional', nome_primeiro_jogador='primeiro', nome_segundo_jogador='segundo')`

Recebe

- uma string indicando a modalidade do jogo a ser criado (com valor padrão `'tradicional'`)
- uma string indicando o nome do primeiro jogador (com valor padrão `'primeiro'`)
- uma string indicando o nome do segundo jogador (com valor padrão `'segundo'`)

e retorna um dicionário de jogo da modalidade indicada, com os jogadores com nomes indicados, sem nenhuma jogada efetuada.

Exemplos:

```
>>> cria_jogo()
{'tamanho': 8, 'primeiro': ['D', 'D', 'D', 'D'], 'segundo': ['D', 'D', 'D', 'D'],
'ocupadas': {}, 'nomes': {'primeiro': 'primeiro', 'segundo': 'segundo'}}

>>> cria_jogo('x-random', 'João', 'Maria')
{'tamanho': 8, 'primeiro': ['B', 'C', 'C', 'C', 'T'],
'segundo': ['B', 'C', 'C', 'C', 'T'], 'ocupadas': {},
'nomes': {'primeiro': 'João', 'segundo': 'Maria'}}
```

3.2 Criando jogadores

Cada jogador será uma função que recebe duas entradas: o dicionário de estado do jogo e uma `string` `'primeiro'` ou `'segundo'` que indica se o jogador é o primeiro ou o segundo no jogo. A função retorna uma `tupla (peça, posição)` indicando a jogada a ser realizada. Idealmente, a jogada realizada deve ser válida, mas essa verificação será feita pela função `roda_jogo`.

Ao criar cada jogador, você pode testá-lo usando as funções `roda_jogo` e `torneio` (esta última sendo mais útil para jogadores que não dependam de interação com o usuário).

Questão 7. Implemente a função `jogador_humano`, que recebe entradas como explicado acima e retorna a jogada do ser humano sentado ao computador (o "usuário"). Para facilitar a vida do usuário, sua função deve perguntar primeiro a peça, e só em seguida perguntar a casa onde a peça será jogada.

Questão 8. Implemente a função `jogador_apressado`, que escolhe a primeira casa válida para jogar uma peça. A ordem em que as peças serão percorridas dentro do `set` não é relevante.

Questão 9. Implemente a função `jogador_roleta`, que escolhe uma jogada válida *aleatoriamente*. Você pode assumir que esta função será chamada *apenas* se houver uma jogada válida para o jogador.

Dica: a função `sample` do módulo `random` recebe dois argumentos, o primeiro sendo uma sequência (tupla, lista ou conjunto) e o segundo um `int`, e retorna uma lista com o tamanho indicado de elementos da sequência dada, escolhidos uniformemente. Caso você receba um aviso `DeprecationWarning`, você pode ignorá-lo com segurança.

Questão 10. Implemente um jogador (com o nome que você quiser) que implementa a chamada "estratégia gulosa": o jogador irá escolher a peça e casa para atacar o máximo possível de casas *ainda não atacadas*. Caso haja mais de uma jogada que ataque o mesmo número máximo, o que o jogador pode escolher qualquer uma.

Questão 11. Implemente um jogador que faça jogadas seguindo alguma outra estratégia que você considerar interessante. Como guia, tente fazer seu jogador ter bom desempenho em torneios contra o `jogador_aleatório` e ser difícil de derrotar na função `roda_jogo` contra o `jogador_humano` (jogando primeiro ou segundo).

Atenção! Seu jogador não deve demorar mais do que alguns poucos segundos para fazer cada jogada.

No dia 20/10, no horário normal de aula, faremos um torneio ao vivo "em sala" entre os jogadores criados pelos alunos como resposta à Questão 11! A participação neste torneio é completamente livre e não conta para a avaliação final de nenhuma forma.

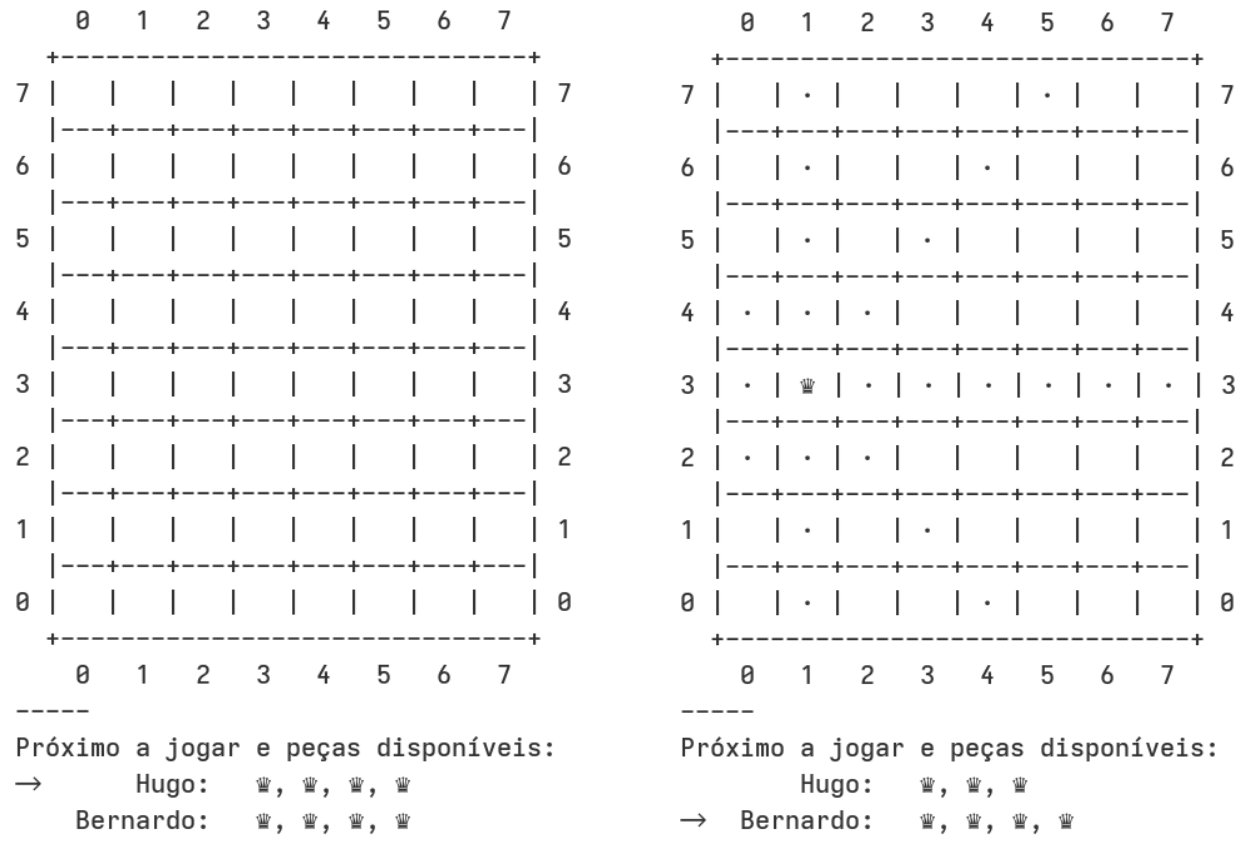

Figura 1: A primeira rodada de um jogo Tradicional.

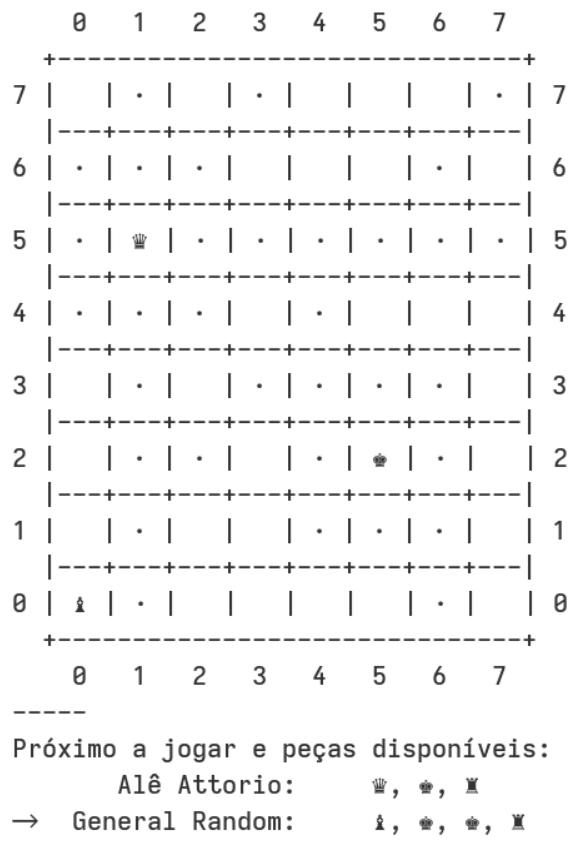

Figura 2: Um jogo X-Random após três rodadas.

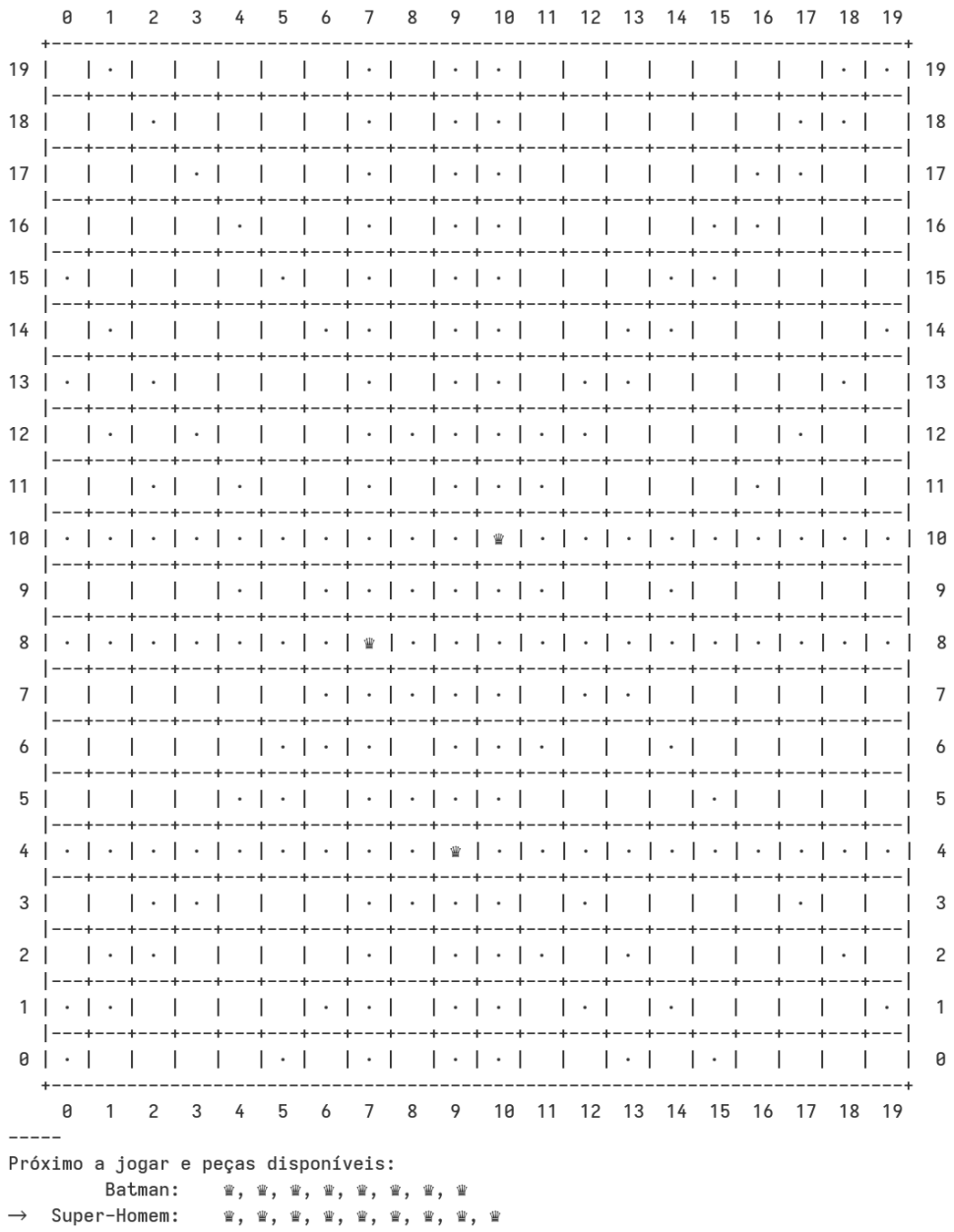

Figura 3: Um jogo Grande após três rodadas.