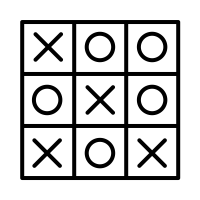
TD3: Noughts and Crosses

# Mô tả

Trò chơi caro (Noughts and Crosses) được mô tả như sau: Có một bảng *m x n*. Hai người chơi lần lượt đánh kí hiệu của riêng mình (X) hoặc (O) vào một ô còn trống của bảng. Người đầu tiên đánh được kí hiệu của mình ở *k* ô liên tiếp theo hàng ngang, cột dọc hoặc đường chéo là người thắng cuộc. Nếu tất cả các ô của bảng đều đã được điền mà không ai thực hiện được nhiệm vụ trên thì



*Trong hình vẽ, m = n = 3. Nếu chọn k = 3, người có kí hiệu X thắng vì đánh được k kí hiệu X liên tiếp trên một đường chéo.*

Trong TD này ta sẽ implement trò chơi này một cách đơn giản, theo các bước sau:

* Thiết lập một class **Board** để mô tả trạng thái của một bàn cờ
* Thiết lập một class **Game** để mô tả trạng thái của ván chơi
* Sau đó, tìm cách để xây dựng chiến thuật khi máy tính chơi với con người sao cho khả năng thắng cao. Ta cũng tìm cách implement chiến thuật sao cho tốn ít thời gian chạy của máy.

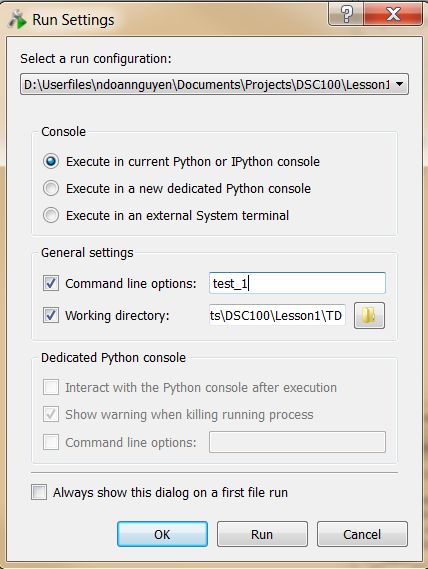
Riêng ở phần cuối, trong phạm vi của TD ta cũng chỉ cần thực hiện cho các trường hợp *m, n, k* nhỏ (khoảng bằng 5).

Bạn cần viết code trong file *NoughtsAndCrosses.py*. Trong file này, một số hàm đã được viết sẵn và một số khác đã được đặt tên sẵn. Bạn không sửa tên các hàm này và không thay đổi code đã viết, trừ trường hợp đổi syntax giữa Python 2 và Python 3.

Bạn có thể viết các hàm phụ khác ngoài các hàm đã yêu cầu trong đề. Bạn có thể sử dụng các hàm từ thư viện khác miễn là hàm đó không trực tiếp trả lời câu hỏi trong bài.

Để test, bạn có thể sử dụng file *TestNoughtsAndCrosses.py*. Bạn không cần thay đổi nội dung file này trừ mục đích đổi syntax để tương thích với Python3.

Để chạy từng test, trong Spyder, bạn chọn Run -> Configure và gõ tên test tương ứng với bài tập (test\_1, test\_2, test\_3, …)



Nếu code viết đúng, khi chạy code bạn sẽ nhìn thấy dòng “Test … OK” và bàn cờ tương ứng. Nếu sai, bạn sẽ nhìn thấy thông báo lỗi của Python (nếu lỗi nằm ở syntax), hoặc của test (nếu implement sai hoặc đôi khi cả lỗi syntax).

Bạn có thể đọc code của từng test trong các hàm test1(), test2() để hiểu hơn ở mỗi bài tập, các hàm cần viết có ý nghĩa như thế nào và kết quả trả lại (return) cần có dạng như thế nào.

# Phần 1. Mô tả bàn cờ - Class Board

Một bàn cờ là một hình chữ nhật được chia thành *m x n* ô bằng nhau. Toạ độ được đánh số như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| (0,0) | (0,1) | (0,2) | (0,3) |
| (1,0) | (1,1) | (1,2) | (1,3) |
| (2,0) | (2,1) | (2,2) | (2,3) |
| (3,0) | (3,1) | (3,2) | (3,3) |

theo quy ước: chỉ số hàng viết trước, chỉ số cột viết sau.

Ta mô tả bàn cờ này bằng class Board. Một instance của class Board, tức một bàn cờ, được xác định bởi chiều cao, chiều rộng, và trạng thái của tất cả các ô của nó (đã được đánh dấu hay chưa). Ban đầu, tất cả các ô của bàn cờ đều trống. Do đó, hàm \_\_init\_\_ trong class được viết như sau:

**def \_\_init\_\_(self, height, width):**

**"""**

**Create a board with defined height and width**

**Fill 0 (empty) to every cell of the board**

**"""**

**self.\_\_height = height**

**self.\_\_width = width**

**self.\_\_cells = []**

**for i in range(height):**

**self.\_\_cells.append([])**

**for j in range(width):**

**self.\_\_cells[i].append(Board.EMPTY)**

Theo đó, tất cả các ô của bàn cờ sẽ được biểu diễn bởi một list self.\_\_cells của *m* list con của *n* số nguyên (int). Trong đó trạng thái trống của một ô sẽ được biểu diễn bởi số nguyên 0, tức hằng số EMPTY của class. Nếu sau này ô (i, j) được đánh bới người chơi 1 (kí hiệu là X) hay 2 (kí hiệu là O), self.\_\_cells[i][j]sẽ mang giá trị tương ứng 1 (X) hoặc 2 (O).

Hàm draw() cho phép phác hoạ trạng thái của bàn cờ. Hãy thử run test0 của TestNoughtsAndCrosses.py, bạn sẽ được kết quả sau (mô tả một bàn cờ 5 hàng, 5 cột, các ô đều trống.

-----------

| | | | | |

-----------

| | | | | |

-----------

| | | | | |

-----------

| | | | | |

-----------

| | | | | |

-----------

Test 0 OK

Bây giờ ta viết code mô tả bàn cờ.

## Bài 1. Các getter, setter cơ bản của Board

Hãy viết các hàm sau trong class Board:

getHeight() : trả lại chiều cao của bảng

getWidth() : trả lại chiều rộng của bảng

getBoardStatus() : trả lại trạng thái của tất cả các ô, tức là một list *m* list con của *n* số nguyên 0, 1 hoặc 2.

setBoardStatus(cells) : nhận tham số cells là một list các list các số nguyên và gán nó cho self.\_\_cells

## Bài 2. Làm việc với từng ô của bàn cờ

Ta sẽ biểu diễn mỗi ô của bàn cờ bằng toạ độ của nó, một tuple (i, j) trong Python, trong đó i là chỉ số hàng, j là chỉ số cột.

1. Hãy viết các hàm sau:

getCellStatus(cell) : nhận tham số cell là toạ độ của ô và trả lại trạng thái của ô đó (một số nguyên 0, 1 hay 2)

isEmptyCell(cell) : nhận tham số cell là toạ độ của ô và trả lại True nếu ô đó đang trống (tức có trạng thái 0) hay False nếu ô đó đã có người đánh (có trạng thái 1 hoặc 2). Lời khuyên: thay vì dùng 0, 1, 2, bạn nên dùng các hằng số được đặt tên theo ý nghĩa để code được rõ ràng, ví dụ 0 = EMPTY, 1 = X hay PLAYER1, 2 = O hay PLAYER2.

mark(value, cell) : nhận tham số cell là toạ độ của ô, value là một giá trị EMPTY = 0, X = 1 hay O = 2, và gán value cho trạng thái của ô đó.

1. Hãy viết hàm:

getEmptyCells() : trả lại list toạ độ tất cả các ô trống trên bàn cờ. Kết quả cần có dạng một list các tuple, ví dụ [(1, 2), (1, 3)]

## Bài 3. Tìm các lân cận của một ô trên bàn cờ

Một ô (x, y) trên bàn cờ được gọi là lân cận của ô (i, j) nếu toạ độ của nó khác (i, j), x nằm trong [i-1, i+1] và y nằm trong [j-1, j+1]. Mỗi ô trên bàn cờ có tối thiểu 3 lân cận (trường hợp của các ô ở góc), tối đa 8 lân cận (trường hợp của các ô không ở biên).

Viết hàm getNeighbors(cell)nhận tham số là toạ độ của một ô trên bàn cờ, và trả lại tất cả các lân cận của nó dưới dạng một list các tuple toạ độ.

# Phần 2. Mô tả ván chơi - Class Game

Một ván chơi được tiến hành như sau: Hai người chơi 1 (X) và 2 (O) thống nhất về kích thước bàn cờ, luật chơi (tức về *m, n, k*), và ai là người đi trước. Sau đó họ tiến hành đánh cờ lần lượt vào các ô trống cho đến khi xác định được người chiến thắng hoặc khi mọi ô của bàn cờ được đánh hết.

Như vậy, các tham số cần cho **thời điểm ban đầu:**

1. *m, n, k* (khác nhau với các ván chơi khác nhau)
2. Người đánh trước là ai.

Ở **mỗi thời điểm (bất kì)**, trạng thái của một ván chơi sẽ hoàn toàn được xác định bởi các thông tin sau:

1. Trạng thái của bàn cờ như thế nào (ô nào đang trống hay đang có kí hiệu gì)
2. Trò chơi đã kết thúc chưa.
3. Nếu trò chơi đã kết thúc, ai là người thắng, hay kết quả là hoà.
4. Nếu trò chơi chưa kết thúc, ai có quyền đánh ở nước tiếp theo.

Ta diễn dịch các thông tin này thành các attribute của các instance trong class **Game**. Ta viết hàm \_\_init\_\_ như sau.

**def \_\_init\_\_(self, height, width, firstTurn, winNumber):**

**"""**

**Begin a game and define which attributes are necessary to describe the game status at some moment**

**"""**

**self.\_\_winNumber = winNumber**

**self.\_\_board = Board(height, width)**

**self.\_\_turn = firstTurn**

**self.\_\_status = Game.ACTIVE**

**self.\_\_winner = Game.NOWINNER**

Các tham số cần cho **thời điểm ban đầu** là *m, n, k, người đi trước* được chuyển thành height, width, winNumber, firstTurn trong phần tham số của \_\_init\_\_.

Các tham số cần cho **mỗi thời điểm** ứng với các điểm 1, 2, 3, 4 nêu trên được mô tả thành các attribute \_\_board, \_\_status, \_\_winner, \_\_turn của class. Trong đó, \_\_board thuộc class Board, ban đầu là một bàn cờ trống; \_\_turn nhận giá trị 1 (X) hoặc 2 (O), ban đầu nhận giá trị firstTurn được cung cấp từ tham số; \_\_status nhận giá trị 1 (ACTIVE) hoặc 0 (INACTIVE), ban đầu ACTIVE; \_\_winner nhận giá trị 0 (NOWINNER, chưa có người thắng hoặc kết quả hoà), 1 (X) hoặc 2 (O), ban đầu là NOWINNER.

*k* sẽ được sử dụng về sau, do đó ta cũng lưu *k* trong attribute \_\_winNumber. *m, n* cũng được sử dụng về sau nhưng đã được lưu trong \_\_board (có thể gọi bằng \_\_board.getHeight(), \_\_board.getWidth()) nên không cần lưu bằng attribute riêng. Nếu cần dùng lại tham số firstTurn về sau, bạn có thể thêm dòng self.\_\_firstTurn = firstTurn vào hàm \_\_init\_\_ theo ý muốn.

Một hàm draw đã được viết sẵn để phác hoạ bàn cờ (tương tự phần 1).

Bây giờ ta viết phần code cần cho một ván chơi.

## Bài 4. Các getter, setter cơ bản của Game

Trong class Game:

1. Hãy viết hàm getBoard() trả lại một instance thuộc class Board mô tả bàn cờ tại một thời điểm bất kì.
2. Hãy viết hàm isActive() trả lại True nếu ván chơi chưa kết thúc và False nếu ván chơi đã kết thúc.
3. Hãy viết hàm getTurn() trả lại số nguyên 1 (X) hoặc 2 (O) cho biết ai có quyền đi nước tiếp theo nếu trò chơi vẫn tiếp diễn.
4. Hãy viết hàm getWinner() trả lại số nguyên 1 (X), 2 (O) cho biết người thắng, hoặc 0 (NOWINNER) nếu chưa có người thắng hay kết quả hoà.
5. Hãy viết hàm deactivate() để chuyển trạng thái của trò chơi thành kết thúc (INACTIVE). Sau khi thực hiện hàm này, nếu gọi lại isActive() thì kết quả là False. Bạn có thể raise error nếu trò chơi đã kết thúc mà hàm deactivate() vẫn được gọi (nếu muốn).
6. Hãy viết hàm activate() để chuyển trạng thái của trò chơi thành tiếp diễn (ACTIVE). Sau khi thực hiện hàm này, nếu gọi lại isActive() thì kết quả là True. Bạn có thể raise error nếu trò chơi đang tiếp diễn mà hàm activate() vẫn được gọi.
7. Hãy viết hàm switchTurn() để gán cho attribute \_\_turn giá trị X (1) nếu đang là O (2) và ngược lại.
8. Hãy viết hàm declareWinner(player) để tuyên bố người chơi player đã thắng (tức gán player cho \_\_winner).

## Bài 5. Đánh cờ

Bây giờ ta viết hàm mark biểu diễn một nước cờ. Giả sử người chơi player muốn đánh kí hiệu của mình vào ô có toạ độ là cell (nhắc lại rằng toạ độ một ô được biểu diễn bởi 1 tuple (i, j) trong Python).

Hãy viết hàm mark(player, cell) lấy tham số là player (người chơi X hoặc O) và cell (toạ độ của ô muốn đánh), kiểm tra xem player có quyền đánh vào ô cell không. Nếu có, đánh kí hiệu của player vào ô cell của bàn cờ (bằng cách dùng hàm mark của class Board đã viết ở bài 2). Nếu không có quyền, hàm không làm gì cả hoặc báo lỗi. Bạn có thể dùng class GameError để báo lỗi nếu muốn.

*Gợi ý: Điều kiện của player, cell và các attribute của ván cờ hiện tại là gì để player có quyền đánh vào ô cell?*

Tới đây, hàm mark vẫn chưa hoàn chỉnh. Ta sẽ cập nhật hàm mark vừa viết trong các bài tập sau.

## Bài 6. Người chơi đã thắng?

Giả sử người chơi player vừa đánh vào ô cell. Nước đánh này có thể làm cho người chơi player thắng cuộc nếu nó tạo được với các ô đã đánh trước đó *k* kí hiệu liên tiếp trên một hàng ngang, cột dọc hoặc đường chéo.

Hãy viết hàm isVictoryCell(cell)nhận tham số là toạ độ cell của một ô và trả lại True nếu nó làm cho người đánh tại ô đó thắng, và False nếu nó chưa làm cho người đánh tại ô đó thắng.

## Bài 7. Bàn cờ đã được đánh hết?

Trong class Game, hãy viết hàm isFull()trả lại True nếu tất cả các ô của bàn cờ đã được đánh hết và False nếu chưa.

(Lưu ý rằng hàm này cũng có thể được viết trong class Board thay vì class Game, vì nó chỉ liên quan đến trạng thái của bàn cờ thay vì ván chơi.)

## Bài 8. Xử lí sau nước đánh

Ngay sau khi người chơi đánh xong, máy tính sẽ thực hiện các thao tác sau:

* Kiểm tra xem nước đánh đó có giúp người chơi thắng không (bằng cách gọi hàm isVictoryCell đã viết ở bài 6) hoặc bàn cờ đã được đánh hết chưa (dùng hàm isFull đã viết ở bài 7).
  + Nếu người chơi thắng, máy sẽ tuyên bố người thắng cuộc (hàm declareWinner(player) đã viết ở bài 4), chuyển trạng thái ván chơi về kết thúc (hàm deactivate()đã viết ở bài 4).
  + Nếu không, nếu bàn cờ đã đầy, máy không tuyên bố người thắng cuộc nhưng chuyển trạng thái ván chơi về kết thúc (hàm deactivate()đã viết ở bài 4).
* Nếu cả trường hợp trên đều không xảy ra, máy sẽ trao quyền đi tiếp cho người tiếp theo (hàm switchTurn()đã viết ở bài 4).

Hãy bổ sung các thao tác này vào cuối hàm mark(player, cell) để hoàn thiện hàm mark.

**Tự test những gì đã viết**

Đến đây việc implement luật chơi vào class Game đã kết thúc. Bạn có thể tự test một ván chơi đơn giản như sau:

X = 1

O = 2

game = Game(5, 5, O, 4)

game.draw()

game.mark(O, (2, 1))

game.draw()

game.mark(X, (2, 2))

game.draw()

game.mark(O, (1, 2))

game.draw()

game.mark(X, (0, 3))

game.draw()

game.mark(O, (1, 1))

game.draw()

…

(Thật ra, chỉ cần gọi game.draw() khi cần thiết)

# Phần 3. Máy tính học chơi cờ - Thuật toán thứ nhất

Ta tiếp tục làm việc với class Game. Thay vì kí hiệu 2 người chơi là X, O ta sẽ xem 1 là con người (HUMAN) và 2 là máy tính (MACHINE). Các hằng số này đã được định nghĩa trong class Game.

Giả sử ván cờ đang ở một trạng thái nhất định, chưa kết thúc và MACHINE sẽ có quyền đánh tiếp. Việc tiếp theo của MACHINE là phải quyết định nên đánh vào ô nào để khả năng thắng lớn nhất.

Chiến thuật của MACHINE như sau:

**Thuật toán 1**

**Input: Trạng thái hiện tại của ván chơi (tức instant thuộc class Game đang chứa hàm implement thuật toán)**

**Output: Toạ độ (i, j) “tốt nhất” để thắng**

* Đối với từng ô (i, j) trong tất cả các ô trống của bàn cờ:
  + Tạo ra N (= 10, 100, 1000, …) bản sao như trạng thái hiện tại của ván chơi.
  + Với mỗi bản sao:
    - Đánh vào ô (i, j)
    - Tiếp tục ván chơi một cách ngẫu nhiên (đóng vai HUMAN đánh vào một ô trống ngẫu nhiên, rồi trở lại vai MACHINE đánh vào một ô trống ngẫu nhiên khác…) cho đến khi ván chơi kết thúc
  + Tính hiệu suất chiến thắng của việc đánh vào (i, j) trên tập hợp tất cả các bản sao đã tạo ra. Hiệu suất có thể được tính như sau:

H(i, j) = Số ván thắng – số ván thua

(chú ý số ván thắng + hoà + thua = N, nên )

* Sau khi có tất cả , chọn làm lớn nhất.

Ta sẽ viết code cho phần này.

## Bài 9. Đánh giá nước vừa đi

1. Trong class Board (không phải class Game), hãy viết hàm getARandomEmptyCell() trả lại toạ độ của một ô đang trống một cách ngẫu nhiên đều (các ô trống có xác suất được chọn như nhau). Nhắc lại toạ độ ô là một tuple của Python.
2. Giả sử trò chơi đang ở trạng thái chưa kết thúc và player là người vừa đánh nước gần nhất. Trong class Game, hãy viết hàm evaluateLastStepRandomly(player)thực hiện việc cho 2 người chơi chơi ngẫu nhiên bằng cách liên tục chọn một ô trống ngẫu nhiên (dùng hàm đã viết ở phần 1.) và lần lượt đánh trên chính ván chơi đó cho đến khi ván chơi kết thúc, sau đó trả lại 1 nếu player thắng, -1 nếu player thua và 0 nếu hoà.

Hàm này tương đương việc đánh giá một cách ngẫu nhiên nước vừa đi của player.

## Bài 10. Tạo một bản sao của game hiện tại

Trong class Game, hãy viết hàm generateGameCopy() trả lại một instance mới thuộc class Game có trạng thái (tức giá trị của mọi attributes) giống hệt trạng thái hiện tại của game đang chơi.

*Lưu ý: Cẩn thận với attribute \_\_board. Tránh sử dụng chung bàn cờ với game chính.*

## Bài 11. Cho máy tính tự chơi với chính nó *N* lần.

Ván chơi đang ở trạng thái chưa kết thúc và player sẽ là người đánh nước tiếp theo. Trong class Game, hãy viết hàm algo1(player) implement **thuật toán 1**, nhận tham số là player và thực hiện các bước của thuật toán. Nhắc lại, các bước đó là:

* Tìm tất cả các ô trống đang có trên bàn cờ
* Với mỗi ô trống (i, j):
  + Tạo một vòng lặp *N* lần (*nên bắt đầu với N nhỏ để kiểm tra tính chính xác của code*), ở mỗi lần
    - Tạo ra một bản sao giống hệt trạng thái ván chơi hiện tại.
    - Đánh vào (i, j) trong bản sao vừa tạo ra (*đừng đánh vào game chính*!)
    - Đánh giá nước vừa đi bằng điểm 1, 0 hoặc -1 bằng cách dùng evaluateLastStepRandomly(player) cho bản sao.
  + Tính tổng điểm của tất cả các ván chơi bản sao trong vòng lặp, đó chính là .
* Trả lại (return) một toạ độ (i, j)làm lớn nhất.

**Tự test những gì đã viết**

Thử test với đoạn code sau (thêm vào cuối file NoughtsAndCrosses.py, xoá sau khi test)

if \_\_name\_\_ == "\_\_main\_\_":

HUMAN = 1

MACHINE = 2

game = Game(3, 3, MACHINE, 3)

print game.decideNextStep(MACHINE, Game.algo1)

Trong đoạn code để test trên, một method decideNextStep(game, player, algo) đã được viết sẵn với mục đích sử dụng các thuật toán khác nhau giải quyết bài toán. Ở đây game.decideNextStep(MACHINE, Game.algo1) nghĩa là ta dùng algo1 cho tham số algo của hàm để quyết định nước đi tiếp theo. Một số thuật toán khác (algo2, algo3) sẽ được giới thiệu ở phần sau.

Kết quả test có hợp lí không? Nếu là người chơi được đi trước trong ván cờ 3x3, bạn sẽ đánh vào ô nào, có giống như lựa chọn của máy không?

Đừng quên chạy test11() trong TestNoughtsAndCrosses.py.

# Phần 4. Cải thiện thuật toán

Thử lập một ván chơi trên bàn cờ 5 x 5 với *k* = 4, MACHINE đi trước. Bạn có thể đánh giá độ nhanh của thuật toán bằng đoạn code sau (thêm vào cuối file NoughtsAndCrosses.py, xoá khi test xong.)

import time

if \_\_name\_\_ == "\_\_main\_\_":

HUMAN = 1

MACHINE = 2

game = Game(5, 5, HUMAN, 4)

game.mark(HUMAN, (2, 2))

t = time.time()

print game.decideNextStep(MACHINE, Game.algo1)

print "Algo 1 took ", time.time() - t, " second."

Sau đó tăng số bản sao của game lên mức 1000 để xem thời gian chạy là bao nhiêu (nếu quá 3 phút, tắt và giảm *N* xuống).

Trên thực tế, thuật toán này có tốc độ không tốt.

Ta thay đổi chiến thuật một chút. Ta cảm giác rằng cả HUMAN và MACHINE đều không nên đánh ở những ô quá xa với những ô đã được đánh, do đó trong thuật toán 1, thay vì tìm tất cả các ô trống trên bàn cờ, ta chỉ tìm những ô trống có ít nhất một lân cận đã được đánh.

Ta gọi các ô trống nằm cạnh một ô không trống của bàn cờ là một **ô tiềm năng.**

Ta có thuật toán dưới đây.

**Thuật toán 2**

**Input: Trạng thái hiện tại của ván chơi (tức instant thuộc class Game đang chứa hàm implement thuật toán)**

**Output: Toạ độ (i, j) “tốt nhất” để thắng**

* Đối với từng ô (i, j) trong tất cả các **ô tiềm năng** của bàn cờ:
  + Tạo ra khoảng N = 1000 bản sao như trạng thái hiện tại của ván chơi.
  + Với mỗi bản sao:
    - Đánh vào ô (i, j)
    - Tiếp tục ván chơi một cách ngẫu nhiên (đóng vai HUMAN đánh vào một **ô tiềm năng**, rồi trở lại vai MACHINE đánh vào một **ô tiềm năng** khác…) cho đến khi ván chơi kết thúc
  + Tính hiệu suất chiến thắng của việc đánh vào (i, j) trên tất cả các bản sao đã tạo ra. Hiệu suất có thể được tính như sau:

H(i, j) = Số ván thắng – số ván thua

(chú ý số ván thắng + hoà + thua = N, nên

* Sau khi có tất cả , chọn tối ưu .

## Bài 12. Thuật toán 2.

Viết hàm algo2() implement thuật toán 2.

Chạy lại đoạn code ở đầu phần, thay Game.algo1 bằng Game.algo2.

Giải thích vì sao thời gian chạy của thuật toán 2 trong trường hợp này nhanh (chậm) hơn.

Chạy test12 (trong đó đã so sánh với test11) và giải thích sự khác biệt tương tự.

## Bài 13. Thuật toán 3

Viết hàm algo3() implement thuật toán 2 ở dạng cho nhiều CPU core chạy đồng thời (dùng multiprocessing.Pool). Mỗi pool đảm nhận một ô trống (là lân cận của một ô đã được đánh).

Chạy test13 (trong đó đã so sánh với test11). Thời gian chạy giảm (tăng) như thế nào?

## Suy nghĩ thêm…

Bạn có thể xây dựng thêm những thuật toán cải thiện hơn để khả năng thắng cao hơn và thời gian chạy nhanh hơn. Thử implement algo4, 5, 6… nếu có ý tưởng.

--------------Hết---------------