BTTH03: Tìm kiếm đối kháng

Môn: Trí tuệ nhân tạo

==========

Ngày 15 tháng 5 năm 2017

1 Hướng dẫn tổng thể

Trong bài này, bạn cũng sẽ cài đặt các thuật toán tìm kiếm để giúp Pacman đi trong mê cung của mình, nhưng bạn sẽ có một số cải tiến để Pacman của bạn có thể đối kháng với các đối thủ khác, cụ thể ở đây là ma. Mục tiêu của đồ án này, bạn sẽ thực hiện cài đặt các thuật toán minimax và expectimax cùng với các hàm đánh giá. Đầu tiên, bạn download khung chương trình của bài tập này ở đây. Sau khi giải nén ra thư mục `multiagent`, bạn mở cmd, và thay đổi thư mục hiện tại trên cmd thành thư mục `multiagent` bằng cách: gõ câu lệnh `cd <Đường dẫn đến thư mục multiagent>` (nếu ổ đĩa hiện tại khác ổ đĩa bạn muốn đến thì bạn gõ `cd /d <Đường dẫn đến thư mục multiagent>`). Kế đến, bạn có thể test thử chương trình với một số câu lệnh sau [mình có đính kèm file `commands.txt` (nên mở bằng Notepad++ hoặc Sublime); trong file này, đã ghi sẵn các câu lệnh để bạn có thể copy-paste cho nhanh]:

- `python pacman.py`: Câu lệnh này sẽ hiện lên game Pacman để bạn có thể chơi thử.
- `python pacman.py -p ReflexAgent`: Câu lệnh này sẽ hiện lên game Pacman, có sử dụng chức năng ReflexAgent trong file multiAgents.py để chơi game. Phiên bản này thì khi chơi game Pacman khá là kém ngay cả khi dùng với hệ thống có kích thước nhỏ, bạn có thể test thử bằng lệnh: `python pacman.py -p ReflexAgent -l testClassic`.

Trong chương trình mà bạn vừa download về, sẽ có một số hàm chưa được viết, một số hàm được viết nhưng chưa hoàn chỉnh/hiệu quả, và nhiệm vụ của bạn là viết hoàn chỉnh các hàm chưa được viết và cải thiện các hàm chưa hiệu quả. Đề bài sẽ có tất cả 5 câu ứng với 5 yêu cầu lập trình (đề bài sẽ được trình bày cụ thể ở mục 2). Với mỗi câu:

- Đầu tiên, bạn sẽ viết hoàn chỉnh hàm được yêu cầu (trong các file *.py, những chỗ mà bạn cần phải viết code sẽ được ghi chú bằng dòng `"*** YOUR CODE HERE ***"`). Để viết được hàm, có thể bạn sẽ phải đọc các phần code liên quan khác; chiến lược đọc ở đây là: đọc một cách ít nhất có thể, chỉ cần đọc đủ để có thể viết được hàm.
- Kế đến, bạn sẽ chạy các câu lệnh để có thể kiểm tra một cách trực quan phần cài đặt của mình trên game Pacman.
- Cuối cùng, bạn sẽ chạy file `autograder.py` để kiểm tra phần cài đặt của mình với các bộ test khác nhau.

Ghi chú thêm: nếu bạn sử dụng Notepad++ để viết code, bạn có thể chọn `View`, rồi `Function List`; chức năng này sẽ giúp bạn xem một cách tổng thể danh sách các hàm có trong file, cũng như là để di chuyển nhanh chóng đến các hàm.

Nên nhớ mục tiêu chính ở đây là *học, học một cách chân thật*. Bạn có thể thảo luận ý tưởng với bạn khác cũng như là tham khảo các tài liệu, nhưng *code và bài làm phải là của bạn, dựa trên sự hiểu của bạn. Nếu vi phạm thì sẽ bị 0 điểm cho toàn bộ môn học.*

2 Đề bài

2.1 Reflex Agent

Class `ReflexAgent` đã được code sẵn trong file `multiAgents.py`, nhưng khi chơi game, Pacman khá là kém trong việc di chuyển và có thể dễ dàng bị ma tiêu diệt. Bạn có thể chạy các lệnh sau để thấy được vấn đề của Pacman hiện tại.

```
python pacman.py -p ReflexAgent -l testClassic
python pacman.py --frameTime 0 -p ReflexAgent -k 1
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

Chính vì vậy, trong phần này, các bạn sẽ viết code để cải tiến class `ReflexAgent`, cụ thể các bạn sẽ phải hoàn chỉnh hàm `evaluationFunction` trong class `ReflexAgent` của file `multiAgents.py`.

Trong class `ReflexAgent` có hai hàm là `getAction()` và `evaluationFunction()`. Hàm `getAction()` sẽ nhận đầu vào là trạng thái của game hiện tại (gameState) và trả về kết quả là hướng đi tốt nhất cho Pacman có thể đi. Kết quả này được đưa ra dựa trên kết quả của hàm `evaluationFunction()`.

Hàm `evaluationFunction()` nhận đầu vào là trạng thái hiện tại của game và một hành động mà Pacman có thể thực hiện từ trạng thái này. Kết quả trả về sẽ là điểm số **ước lượng** của hành động này, điểm số càng cao thì hành động càng được ưu tiên. Đoạn code có sẵn trong hàm `evaluationFunction()` cung cấp sẵn cho chúng ta cách trích xuất một số thông tin hữu ích từ trạng thái của game. Cụ thể như:

```
// Lấy vị trí của Pacman sau khi Pacman thực hiện action.
newPos = successorGameState.getPacmanPosition()
// Lấy trạng thái mới của game sau khi Pacman thực hiện action.
successorGameState = currentGameState.generatePacmanSuccessor(action)
// Cập nhật phần thức ăn còn lại sau khi Pacman thực hiện action.
newFood = successorGameState.getFood()
// Cập nhật trạng thái mới của ma sau khi Pacman thực hiện action.
newGhostStates = successorGameState.getGhostStates()
```

Bạn sẽ sử dụng những thông tin có sẵn này để tiến hành cài đặt hoàn chỉnh hàm `evaluationFunction()`.

Sau khi cài đặt xong, bạn có thể chạy lệnh:

```
python autograder.py -q q1
```

Lệnh này sẽ thực hiện chạy Pacman trên openClassic layout 10 lần khác nhau. Nếu bạn bị time out trong quá trình chạy, bạn sẽ không thể nào thắng được ma, khi đó chương trình của bạn sẽ nhận được 0 điểm. Bạn sẽ nhận được 1 điểm nếu Pacman thắng ma ít nhất 5 lần, 2 điểm nếu Pacman thắng ma tất cả 10 lần. Bạn sẽ nhận thêm được 1 điểm, nếu điểm trung bình sau các lần chơi game của bạn lớn hơn 500 điểm. Bạn sẽ nhận được 2 điểm, nếu điểm trung bình sau các lần chơi game lớn hơn 1000 điểm.

Bạn có thể sử dụng câu lệnh `python autograder.py -q q1 -no-graphics` để thực hiện đánh giá điểm một cách nhanh chóng hơn. Câu lệnh này sẽ không hiển thị lên màn hình quá trình chơi game của Pacman.

2.2 Minimax

Trong phần này, bạn sẽ viết code vào hàm `getAction()`, trong class `MinimaxAgent`. Bạn có thể viết thêm các hàm phụ để hỗ trợ cho hàm `getAction()` nếu cần thiết. Thuật toán minimax của bạn thiết kế nên đảm bảo chạy được với số lượng ma là bất kì. Lượt chơi của Pacman và ma luân phiên nhau như sau: Pacman đi, rồi lần lượt từng ma đi, rồi Pacman đi, ...

Khi viết hàm `getAction()` (cũng như là các hàm phụ trợ) trong class `MinimaxAgent`, bạn có thể gọi `self.depth` và `self.evaluationFunction` (`self.depth` và `self.evaluationFunction` được định nghĩa trong class `MultiAgentSearchAgent`, class `MinimaxAgent` kế thừa từ class này):

- `self.depth` cho biết biết độ sâu giới hạn của cây (số bước nhìn xa về tương lai). *Lưu ý: một đơn vị độ sâu ở đây gồm một lần đi của Pacman và tất cả các ma sau đó; như vậy, độ sâu bằng 2 nghĩa là: Pacman đi, tất cả các ma lần lượt đi, Pacman đi, tất cả các ma lần lượt đi.*

- `self.evaluationFunction` nhận đầu vào là một trạng thái mà không phải là trạng thái kết thúc game và trả về giá trị ước lượng của trạng thái đó. `self.evaluationFunction` mặc định là `scoreEvaluationFunction`: nhận đầu vào là một trạng thái và trả về giá trị ước lượng là điểm số của game tại trạng thái đó. Trong câu này, bạn chỉ đơn giản là sử dụng hàm ước lượng mặc định đã được viết sẵn này. (Ở đây, hàm ước lượng hơi khác với hàm ước lượng của Reflex Agent mà bạn viết ở câu trước: ở câu trước, hàm ước lượng nhận đầu vào là trạng thái và hành động; còn ở đây, hàm ước lượng nhận đầu vào là trạng thái).

Để debug, kiểm tra code và đánh giá chương trình của mình, bạn có thể chạy chương trình bằng lệnh `python autograder.py -q q2`.

Bạn có thể chạy lệnh: `python autograder.py -q q2 -no-graphics` để chạy chương trình mà không cần hiển thị graphics.

Một số ý khác:

- Cài đặt đúng của minimax trong trường hợp này là Pacman sẽ bị ma tiêu diệt trong một số bộ test. Điều này là đúng, không phải do thuật toán của bạn, vì ở đây minimax chưa được tối ưu.

- Giá trị minimax của trạng thái khởi tạo trong minimaxClassic layout là 9, 8, 7, -492 tương ứng với các độ sâu là 1, 2, 3 và 4. Lưu ý rằng thuật toán của bạn sẽ thường thắng ma, với cả độ sâu là 4. Các bạn có

thể test thử bằng lệnh sau: `python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4`.

- Khi bạn chạy trên các layout lớn hơn như openClassic và mediumClassic, bởi lệnh:

  `python pacman.py -p MinimaxAgent -l openClassic` hoặc

  `python pacman.py -p MinimaxAgent -l mediumClassic`

  Bạn sẽ nhận thấy rằng Pacman không thể bị ma tiêu diệt, nhưng cũng không thể nào chiến thắng được ma. Bạn sẽ khắc phục vấn đề này trong câu hỏi 5 của bài tập này.

- Lệnh `gameState.generateSuccessor(agentIndex, action)`: // trả về trạng thái successor của game sau khi agent thực hiện hành động.

- Lệnh `gameState.getNumAgents()`: // trả về tổng số agent trong game.

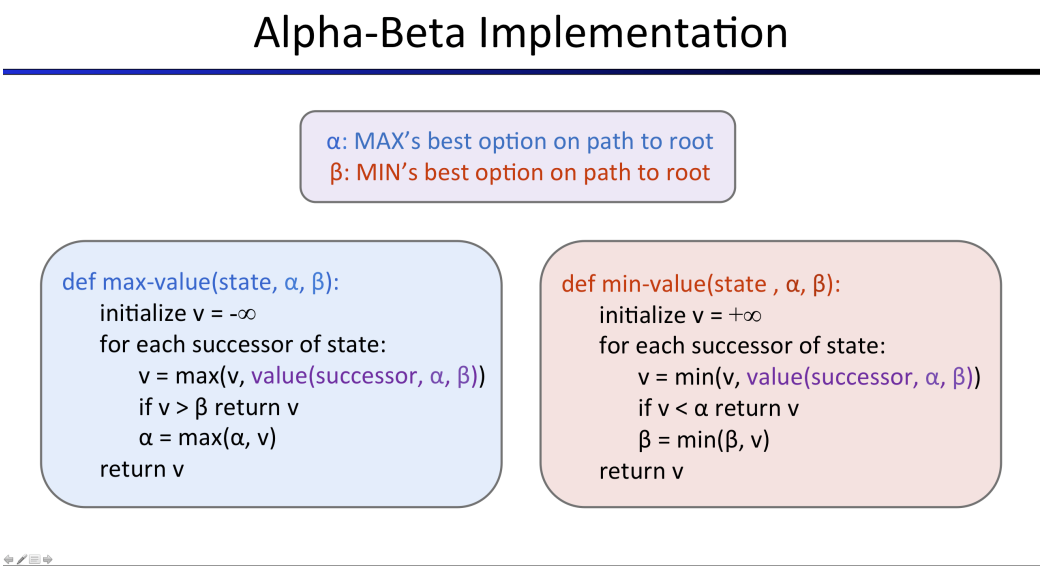
- Lệnh `gameState.getLegalActions(agentIndex)`: // Trả về các hành động mà agent có thể thực hiện.

2.3 Alpha-Beta Pruning

Trong phần này, bạn sẽ viết code vào hàm `getAction()`, trong class `AlphaBetaAgent`. Bạn có thể viết thêm các hàm phụ để hỗ trợ cho hàm `getAction()` nếu cần thiết. Sau khi cài đặt thành công, bạn sẽ thấy tốc độ chạy của alpha-beta với độ sâu là 3, sẽ nhanh hơn minimax với độ sâu là 2. Bạn có thể quan sát điều này bằng cách chạy lệnh sau:

`python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic`

Alpha-Beta Pruning là thuật toán cải tiến của Minimax, bạn có thể dùng lại các thông tin trong class `MinimaxAgent` để phục vụ cho quá trình cài đặt chương trình của mình. Cụ thể, bạn có thể xem sự thay đổi của các hàm `max-value()` và `min-value()` thông qua mô tả thuật giả của các hàm này như hình sau:



Hình 1: Thuật giả Alpha-Beta

Để debug, kiểm tra code và đánh giá chương trình của mình, bạn có thể chạy chương trình bằng lệnh `python autograder.py -q q3`.

Bạn có thể chạy lệnh: `python autograder.py -q q3 -no-graphics` để chạy chương trình mà không cần hiển thị graphics.

Thực hiện đúng alpha-beta pruning, Pacman sẽ có thể bị ma tiêu diệt trong một số bộ test. Đây không phải lá vấn đề do chương trình của bạn, mà là do thuật toán alpha-beta chưa được tối ưu.

2.4 Expectimax

Cả hai thuật toán Minimax và Alpha-Beta đều dự đoán hành động của đối thủ bằng cách giả định là đối thủ chơi tối ưu. Trên thực tế, không phải lúc nào đối thủ cũng chơi tối ưu, vậy nếu đối thủ chơi không tối ưu, ta sẽ dự đoán hành động của đối thủ bằng cách nào? Trong phần này bạn sẽ tiến hành cài đặt thuật toán Expectimax để giải quyết vấn đề này. Bạn sẽ viết code vào hàm `getAction()`, trong class `ExpectimaxAgent`. Bạn có thể viết thêm các hàm phụ để hỗ trợ cho hàm `getAction()` nếu cần thiết.

Các ma sẽ được random ngẫu nhiên, vì vậy hành động của ma lúc này có thể là hành động tối ưu, có thể là hành động không tối ưu. Do đó, khi

tính giá trị của trạng thái dưới sự kiểm soát của đối thủ ta sẽ xét trường hợp trung bình (tính trung bình - expectation).

Lưu ý, khi tính giá trị trung bình trên python, bạn phải đảm bảo những biến truyền vô kiểu float, nếu không chương trình của bạn có thể sẽ bị tính sai. Ví dụ: $1/2 = 0$ (những biến kiểu int), $1.0/2.0 = 0.5$ (những biến kiểu float).

Để debug, kiểm tra code và đánh giá chương trình của mình, bạn có thể chạy chương trình bằng lệnh `python autograder.py -q q4`.

Sau khi cài đặt thành công, bạn có thể quan sát cách Pacman hành động trong thuật toán Expectimax bằng cách chạy lệnh sau:

```
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

Bạn có thể quan sát sự khác nhau giữa AlphaBetaAgent với Expectimax-Agent bằng cách chạy hai lệnh sau:

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3
-q -n 10
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3
-q -n 10
```

2.5 Evaluation Function

3 Nộp bài

Trong thư mục `<MSSV>`, bạn đặt file `multiAgents.py` rồi nén thư mục này lại và nộp ở link trên moodle.

Enjoy searching :-)