인공지능 과제 #2

0. Introduction

인공지능 강의의 두번째 과제는 두개의 독립된, 그러나 같은 알고리즘으로 구현되는 문제로 구성되어 있습니다. 먼저, pacman problem 으로 alpha-beta pruning 에 익숙해지고, 이어서 Othello 를 이용한 대전 게임을 진행하면 순서가 적절할 것입니다.

1. Multi-Agent Pacman

http://ai.berkeley.edu/multiagent.html

1.1 학습목표

수업시간에 배운 adversarial search algorithm 중 alpha-beta pruning 을 구현하여 실제 문제에 적용함으로 프로그래밍 능력의 향상을 도모합니다.

1.2 요구사항

AlphaBetaAgent 에 alpha-beta pruning 을 이용한 에이전트를 구현합니다. 구현된 AlphaBetaAgent 는 여러 ghost 가 있는 경우에도 동작하여야 하므로, 강의시간에 익힌 algorithm 보다 다소 일반화될 필요가 있습니다. 구체적으로, 구현된 minimax tree 는 하나의 max layer 에 대하여 복수의 min layer (ghost 당 하나씩)를 가질 수 있는 점에 유의하기 바랍니다.

구현된 코드는 game tree 를 임의의 depth 까지 확장할 수 있어야 합니다. minimax tree 의 leaf node 의 스코어는 제공된 self.evalutionFunction 을 이용합니다 (defaults to scoreEvaluationFunction). AlphaBetaAgent 는 MultiAgentSearchAgent 를 상속하며, 이를 이용하여 self.depth 와 self.evaluationFunction 을 사용할 수 있습니다. 이 변수들은 command line 에서 전달받으므로, 정확히 참조하여야 함에 주의하시기 바랍니다.

하나의 search ply 는 pacman 이동 한번과 그에 따른 모든 ghost 의 대응이며, 따라서 depth 2 search 는 pacman 과 ghost 가 각각 두번씩 움직이는 것을 뜻합니다.

1.3 실행방법

pacman 은 command line option 에 따라 다양하게 실행할 수 있습니다. 다음은 하나의 예시입니다.

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

평가 실행 명령:

```
python autograder.py -q q3
```

2. Othello

2.1 학습목표

다음 조건을 만족하는 Othello 게임을 위한 adversarial search program 을 작성하는 과제입니다. 알고리즘은 Alpha-Beta pruning 을 하는 MinMax 알고리즘을 사용해야 합니다. 그리고 성능(낮은 time complexity 와 이길 수 있는 전력/evaluation function)이 무척 중요합니다. 추후, 여러분이 작성한 프로그램 상호간 play 를 시켜 순위를 정할 예정입니다.

2.2 요구사항

Othello 규칙은 다음과 같습니다.

- 처음에 판 가운데에 사각형으로 엇갈리게 배치된 돌 4개를 놓고 시작한다.
- 돌은 반드시 상대방 돌을 양쪽에서 포위하여 뒤집을 수 있는 곳에 놓아야 한다.
- 돌을 뒤집을 곳이 없는 경우에는 차례가 자동적으로 상대방에게 넘어가게 된다.
- 아래와 같은 조건에 의해 양쪽 모두 더 이상 돌을 놓을 수 없게 되면 게임이 끝나게 된다.
    - 64개의 돌 모두가 판에 가득 찬 경우 (가장 일반적)

- 어느 한 쪽이 돌을 모두 뒤집은 경우
- 한 차례에 양 쪽 모두 서로 차례를 넘겨야 하는 경우

- 게임이 끝났을 때 돌이 많이 있는 플레이어가 승자가 된다. 만일 돌의 개수가 같을 경우는 무승부가 된다.

프로그램 작성 시 다음에 주의하세요.

- 주어진 python 파일에서 “nextMove” 함수를 구현한다. 이 함수는 다음과 같은 세 개의 파라미터를 필요로 한다.
  - Board state: 다음과 같은 nested list로 현재 보드의 상태를 나타낸다. 초기에는 그림1과 같은 내용을 갖는다.
  - Color: “B”나 “W”로 현재 play하는 색을 나타낸다.
  - Time: 게임이 끝날 때까지 사용할 수 있는 남은 시간을 나타낸다.
    - 한번 play할 때 마다 소요된 시간을 차감하여 계산
    - 다른 사람이 작성한 프로그램과 대전할 때 사용할 예정

주의:

- 각자 nextMove 함수를 작성할 때, 깊이 생각하여 시간적 효율과 evaluation function의 지능을 높이도록 하세요.
- Depth level은 최대 4까지 가능합니다.
- 어떤 형태든 부정행위는 최종 학점이 F로 처리됩니다.

2.3 실행방법

- 주어진 Python 파일

- gamePlay.py: command line을 이용하여 서로 게임을 할 수 있게 하는 파일
  - 다음과 같은 명령에 의해 실행
    - % python gameplay.py [-t<timelimit>] [-v] player1 player2
  - player1.py 과 player2.py 이 nextMove 함수를 포함하는 python file 이라면 v 옵션은 각 play의 보드를 보여줍니다.
- randomPlay.py: 랜덤하게 말을 두게 하는 샘플 play 파일
  - 각자 작성한 프로그램을 이 파일의 play와 대결시켜 볼 수 있도록 제공한 파일
- simpleGreedy.py: evaluation 함수는 있으나 매우 무식하게 두는 샘플 play 파일
  - 각자 작성한 프로그램을 이 파일의 play와 대결시켜 볼 수 있도록 제공한 파일
- 만약, 두 명의 random player가 대결하게 하려면 다음과 같은 명령어를 사용하면 됩니다.
  - % python gameplay.py randomPlay randomPlay
- 만약 simpleGreedy 프로그램과 randomPlay 프로그램이 64초의 시간 제약을 두고 서로 게임하게 하려면 다음과 같은 명령어를 사용하면 됨 (simpleGreedy가 먼저 play 함)
  - % python gameplay.py –t64 -v simpleGreedy randomPlay

```
[
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', 'W', 'B', '.', '.', '.'],
['.', '.', '.', 'B', 'W', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.'],
['.', '.', '.', '.', '.', '.', '.', '.']
]
```

<그림 1>

3. 과제제출

기한: 5 월 4 일

late submission policy: 최대 3 일, 정상 채점 후 하루당 10% 감점

4. 과제문의

클래스넷 및 T813