Project 2 : Multi-Agent Pacman

소개

P2 프로젝트에서는 고스트 또한 고려하여 우리의 agent를 디자인합니다. minimax와 expectimax 탐색 알고리즘을 구현하고 직접 평가함수를 디자인합니다.

코드의 베이스는 이전 프로젝트와 많이 바뀌지는 않았지만 프로젝트1의 파일을 혼합하지 않고 새로이 사용합니다.

P1과 마찬가지로, 자동 채점이 포함되어 있으며, 다음 명령어를 통해 실행시킬 수 있습니다.
python autograder.py

다음과 같이 q2에 대한 특정 문제에 대해서도 채점이 가능합니다.
python autograder.py -q q2

아래 명령어를 통해 특정 테스트를 실행할 수도 있습니다.
python autograder.py -t test_cases/q2/0-small-tree

파일 설명

수정하거나 실행하는 파일들

multiAgents.py	Where all of your multi-agent search agents will reside.
pacman.py	The main file that runs Pacman games. This file also describes a Pacman GameState type, which you will use extensively in this project
game.py	The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
util.py	Useful data structures for implementing search algorithms.

무시해도 되는 파일들

graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents
autograder.py	Project autograder
testParser.py	Parses autograder test and solution files
testClasses.py	General autograding test classes
test_cases/	Directory containing the test cases for each question
multiagentTestClasses.py	Project 2 specific autograding test classes

이 문서는 http://ai.berkeley.edu/multiagent.html 를 번역하여 만들어졌습니다.

Q1 - Reflex Agent

multiAgent.py의 ReflexAgent의 성능을 개선하시오. 주어진 ReflexAgent 예제는 게임 상태에 대한 유용한 정보들을 쿼리하는 방법들의 예시를 제공합니다. agent는 쉽게 간결하게 testClassic 레이아웃을 클리어 해야합니다.

python pacman.py -p ReflexAgent -l testClassic

하나 또는 두 개 이상의 고스트와 함께 mediumClassic 레이아웃에서 RelexAgent를 수행해 해 볼 수 있습니다. (속도를 올리기 위해서는 애니메이션을 끄세요.)

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

옵션 : 기본적으로 귀신은 무작위로 움직입니다. -g DirectionalGhost 옵션을 사용할 경우 좀 더 똑똑한 유령을 사용할 수 있습니다. 랜덤성으로 인해 에이전트가 성능개선의 확인이 어려울 경우 -f 옵션을 사용하여 고정 임의 시드값을 설정해 실행할 수 있습니다. -q 옵션을 통해 그래픽을 끌 수 도 있으며, 이는 게임 재생속도를 빠르게 합니다. 또한 -n 옵션을 사용하여 여러 게임을 연속으로 재생할 수 있습니다.

채점방식 : 여러분의 agent를 OpenClassic 레이아웃에 10번 실행합니다. 시간초과 및 패배할 경우에는 0점을 받게 됩니다. 에이전트가 최소 5회 이상 이길 경우는 1점, 10게임을 모두 이길경우는 2점을 받게 됩니다. 또한 에이전트의 평균 점수가 500점보다 큰 경우에는 1점을 1000점보다 큰 경우에는 2점을 추가로 받게 됩니다. 테스트는 아래 명령어를 통해 실행할 수 있습니다.

python autograder.py -q q1
python autograder.py -q q1 --no-graphics (그래픽 제거)

Q2 - Minimax

본 문제에서는 multiAgent.py 파일 안의 MinimaxAgent 클래스에 적대적 탐색 agent를 작성하게 됩니다. 여러분이 작성한 minimax agent는 고스트의 수에 관계없이 잘 작동하여야 하기에, 이전의 문제에서 본 것 보다 좀 더 일반적인 알고리즘을 작성해야 합니다. 특히 여러분의 minimax tree는 모든 max 레이어에 대해 다양한 min 레이어(각 고스트에 해당하는)를 가질 것 이다.

중요 : 하나의 탐색 플레이는 하나의 팩맨의 이동과 모든 유령의 응답으로 간주됩니다. 따라서 깊이 2의 탐색은 팩맨과 유령이 두 번 이동하는 것을 의미합니다.

채점 : 정확한 수의 게임 상태를 탐색하는지 여부를 확인하기 위해 코드를 검사합니다. 이것은 minimax의 구현에서 버그가 있는지 혹은 제대로 구현되었는지를 감지하는 유일한 방법입니다. 따라서 Autograder는 GameState.generateSuccessor를 몇 번 호출했는지 매우 까다롭게 확인합니다. 채점 방법은 아래와 같습니다.

python autograder.py -q q2

python autograder.py -q q2 --no-graphics

힌트 및 관찰
1. minimax를 올바르게 구현하였다면 여러분의 agent는 일부 테스트 게임에서 패배하게 됩니다. 이는 바람직한 동작으로 패배하더라도 테스트를 통과하게 됩니다.

2. 평가 함수는 이미 작성이 되어 있으며, 변경하면 안됩니다. (self.evaluationFunction) 이 평가함수는 action이 아닌 state를 평가하고 있음을 인지하고 있으셔야 합니다. 이 에이전트는 미래 상태들을 평가하지만 앞서 살펴본 reflex 에이전트는 현재상태에서의 액션을 평가한다는 차이가 있습니다.

3. minimaxClassic 레이아웃의 초기 상태의 미니 맥스 값은 각각 깊이 1, 2, 3 및 4에 대해 9, 8, 7, -492입니다. 미니 맥스 에이전트는 깊이 4 미니 맥스 (minimalmax)를 예측하지 못했지만 종종 승리합니다 (665/1000 게임).

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

4. Pagman은 에이전트 0이며, 에이전트는 에이전트 인덱스를 증가시키는 순서대로 이동합니다.

5. minimax의 모든 상태는 GameState 여야하며 getAction에 전달되거나 GameState.generateSuccessor를 통해 생성됩니다. 이 프로젝트에서는 단순화 된 상태로 추상화하지 않을 것입니다.

6. OpenClassic및 MediumClassic(기본 값)과 같은 더 큰 보드에서 Pacman이 죽지는 않지만 제법 승리하지 못한다는 것을 발견할 수 있습니다. 그는 종종 진전없이 주위를 때려 눕힐 것입니다. 그는 그 점을 먹은 후에 어디로 갈지 모르기 때문에 그것을 먹지 않고 도트 바로 옆에서 쓰레기를 던질 수도 있습니다. 이 문제가 발생해도 걱정하지 마십시오. 문제 5는 이러한 모든 문제를 해결합니다.

7. Pacman은 자신의 죽음이 불가피하다고 생각할 때, 가능한 한 빨리 게임을 끝내려고 노력할 것입니다. 때로는 이것은 무작위로 유령을 사용하는 것은 잘못된 것이지만 minimax 에이전트는 항상 최악의 경우를 가정합니다.

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

이 경우 Pagman이 가장 가까운 귀신을 쫓아내는 이유를 이해했는지 확인하세요.

Q3 - Alpha-Beta Pruning

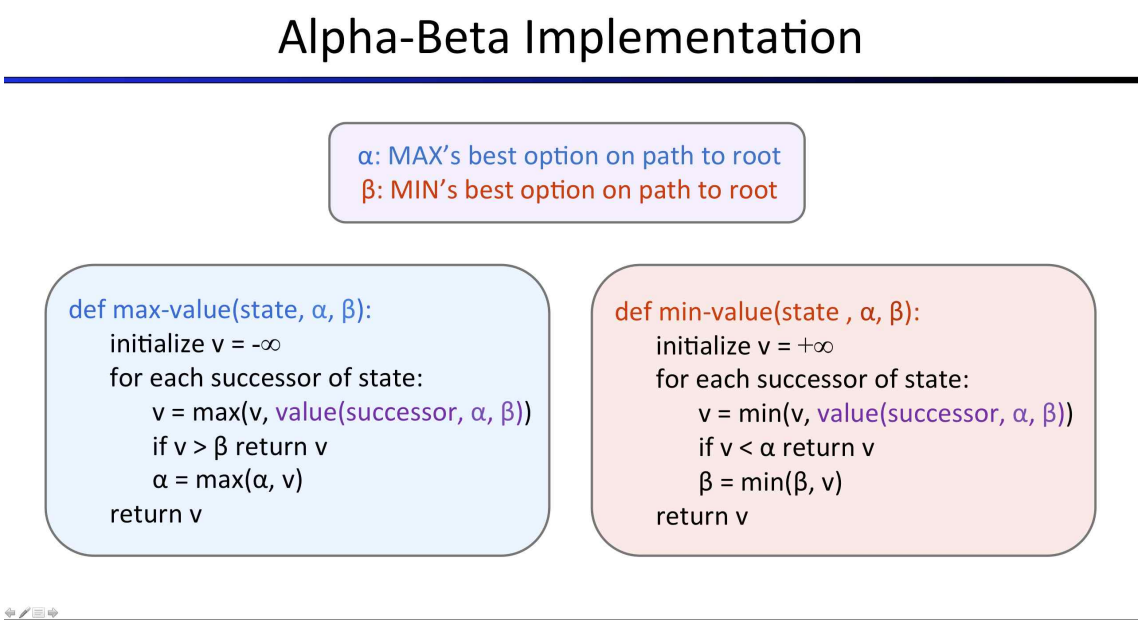
AlphaBetaAgent 에서 alpha-beta 가지치기를 사용하여 minimax 트리 보다 훨씬 효율적인 에이전트를 만듭니다. 여러분의 알고리즘은 강의 의사코드보다 약간 더 일반화 될 것이므로, 과제의 일부는 alpha-beta 가지치기 로직을 다중 최소화 에이전트에 적절하게 확장하는 것입니다.

속도를 올려야 합니다(아마도 깊이 3alpha-beta는 깊이 2minimax만큼 빠르게 실행될 것입니다). 이상적으로, smallClassic의 깊이 3은 이동 당 몇초 또는 그 이상의 속도로 가야 합니다.

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

AlphaBetaAgent minimax 값은 MinimaxAgent minimax 값과 동일해야하지만, 선택한 동작은 서로 다른 묶음 동작으로 인해 다를 수 있습니다. 다시, minimaxClassic 레이아웃의 초기 상태의 minimax 값은 깊이 1, 2, 3 및 4 각각에 대해 9, 8, 7 및 -492입니다.

등급 매기기 : 올바른 상태 수를 탐색하는지 여부를 확인하기 위해 코드를 검사하기 때문에 자식노드들을 재정렬하지 않고 알파 베타 제거를 수행하는 것이 중요합니다.즉, 후행 상태는 항상 GameState.getLegalActions에 의해 반환 된 순서대로 처리되어야합니다. GameState.generateSuccessor를 필요 이상으로 호출하지 마십시오. autograder에 의해 탐험 된 상태집합과 일치하기 위해 임의로 제거해서는 안됩니다.



코드를 테스트하고 디버깅하려면 다음을 실행하십시오.
python autograder.py -q q3
python autograder.py -q q3 --no-graphics
알파 베타 가지치기 기능을 올바르게 구현하면 Pacman이 일부 테스트를 패배하게 됩니다. 이는 문제가되지 않습니다. 올바른 동작이므로 테스트를 통과합니다.

Q4 - Expectimax

미니 맥스 (Minimax)와 알파 베타 (alpha-beta)는 훌륭하지만, 둘 다 최적의 결정을 내리는 적과 싸우고 있다고 가정합니다. 대부분의 사람들이 알 듯 항상 그러하지 않습니다. 따라서 차선 선택을 할 수 있는 에이전트의 확률적 동작 모델링에 유용한 Expectimax에이전트를 구현할 것이다.

이 클래스에서 지금까지 다룬 탐색 및 제약 조건 만족 문제와 마찬가지로, 이러한 알고리즘의 장점은 일반적인 적용성입니다. 개발을 촉진하기 위해 generic tree에 기초한 몇가지 테스트 케이스를 제공해 드렸습니다. 다음 명령을 사용하여 작은 게임 트리에서 구현을 디버그 할 수 있습니다.

python autograder.py -q q4

이러한 작고 관리 가능한 테스트 케이스에 대한 디버깅을 수행하는 것이 좋으며 버그를 신속하게 발견하는 데 도움이 될 것입니다. 평균을 계산할 때는 float을 사용해야 합니다. Python의 정수 분할을 사용하면 1.0V/2.0=0.5인 float을 사용하는 경우와는 달리 1/2=0이 되도록 자를 수 있습니다.

여러분의 알고리즘이 small tree에서 잘 작동하면 pacman의 성공적인 움직임을 확인할 수 있습니다. 무작위로 움직이는 유령은 minimax 알고리즘으로 모델링 하기에 적합하지 않을 수 도 있습니다. ExpectimaxAgent는 더이상 유령의 모든 행동에 시간을 쓰지않고, 유령이 어떻게 행동하는지에 대한 당신의 모델에 따라 예측하게 됩니다. 코드를 단순화하기 위해 getLegalActions 중 임의적으로 임의로 선택하는 적들에 대해서만 실행한다고 가정합니다.

ExpectimaxAgent가 Pacman에서 어떻게 작동하는지 보려면 다음을 실행하십시오.
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3

유령과 가까운 구역에서 좀 더 무심한 접근법을 관찰해야합니다. 특히 Pacman이 갇혀있을 수는 있지만 몇 가지 음식을 얻기 위해 탈출 할 수 있다고 생각하면 적어도 시도 할 것입니다. 다음 두 시나리오의 결과를 조사하십시오.

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

AlphaBetaAgent가 항상 패배하는 동안 ExpectimaxAgent는 약 절반의 시간만에 승리합니다. 미니 맥스의 경우와 다른 점을 이해하십시오.

expectimax를 올바르게 구현하면 Pacman이 일부 테스트를 패배하게 됩니다. 이는 문제가되지 않습니다. 올바른 동작이므로 테스트를 통과합니다.

Q5 - Evaluation Function

제공된 함수 betterEvaluationFunction에서 팩맨에 대한 더 나은 평가 함수를 작성하십시오. 평가함수는 relfex agent의 평가함수가 action을 평가하는것과 달리 state를 평가해야 합니다. 여러분의 프로젝트 탐색 코드를 포함하여 평가를 위해 사용할 수 있습니다. 2의 깊이 탐색시, 당신의 평가 함수는 하나의 무작위로 움직이는 고스트를 포함한 smallClassic 레이아웃을 클리어하고 합리적인 속도로 실행해야 합니다.

python autograder.py -q q5

채점 : 자동 분류기는 smallClassic 레이아웃에서 에이전트를 10 번 실행합니다. 다음과 같은 방법으로 포인트를 평가 함수에 지정합니다.

- 자동 기록기를 시간 초과하지 않고 적어도 한 번 이기면 1 점을받습니다. 이 기준을 만족시키지 못하는 에

이전트는 0 점을 받습니다.
- 최소 5회 이상 이긴 경우+1,10회 이긴 경우+2
- 평균 점수가 500 점 이상인 경우 +1, 평균 점수가 1000 점 이상인 경우 +2 (게임 분실시 점수 포함)
- 자동 기록 장치에서 게임의 평균 소요 시간이 30 초 미만인 경우 +1하십시오. Autograder는 EC2에서 실행되므로이 컴퓨터의 리소스는 상당하지만 개인용 컴퓨터의 성능은 훨씬 낮을 수 있습니다 (넷 북) 또는 훨씬 더 성능이 좋은 (게임 장비).
- 평균 점수 및 계산 시간에 대한 추가 포인트는 최소한 5 회 우승 한 경우에만 수여됩니다.