인공지능 (2023년도 1학기)

과제 #2 – PacMan Programming

본 과제에서는 수업시간에 배운 A* search와 Alpha-Beta Pruning, Q-Learning 알고리즘을 이용하여 PacMan 게임을 잘 수행하는 agent를 구현해 봅니다. 제공되는 압축파일(23AI_hw2_code.zip)을 풀면, `search`와 `multiagent`, `reinforcement` 3개의 디렉토리들을 확인할 수 있습니다. 각 디렉토리에 대해서 A* search와 Alpha-Beta Pruning, Q-Learning 문제를 요구사항에 구현하면 됩니다.

과제 진행시 주의 사항은 다음과 같습니다.

1. 구현해야 하는 파일과 코드가 아닌 게임 그래픽 관련된 코드나 다른 코드들은 수정하지 않아야 합니다.
2. 모든 채점은 `autograder.py` 파일을 통해서 이루어집니다.
3. 과제 제출
   A. 기한: **5월 29일 23:59**까지
   B. 방법: LMS 인공지능 과목의 레포트 제출에서 **구현이 포함되어 있는 파일들을 하나의 zip 파일로 압축해서 제출하시기 바랍니다.** (파일 이름 예시, 홍길동_20000000.zip)
4. **주의 사항**
   A. zip 형식 이외의 다른 압축형식으로 제출했거나, zip 형식 파일이어도 윈도우 PC에서 압축이 풀리지 않은 파일들은 채점에서 제외됩니다.
   B. 다른 사람의 코드를 그대로 사용한 것이 적발되면 보여준 사람, 베낀 사람 모두 0점 처리합니다.
   C. 인터넷에 공개된 코드나 로직을 베껴서 사용하지 마세요. 과제에 사용한 것이 적발되면 해당과제는 0점 처리합니다. 여러분들이 인터넷에서 찾을 수 있을만한 자료들은 이미 조교들이 모두 확보하고 있다고 생각하시면 됩니다. 스스로의 힘으로 과제를 풀어보세요.

1. A* search (3 points)

A* search 구현에 필요한 코드는 `search/search.py`의 `aStarSearch` 함수에 작성하시오. A*는 heuristic 함수를 인자로 받는데, heuristic은 search 문제의 상태(`main` 인자)와 문제 그 자체(`reference` 정보) 2개의 인자를 사용한다.

작성된 코드는 다음과 같은 명령어를 통해서 테스트할 수 있다. (`manhattanHeuristic`은 이미 `searchAgents.py` 파일에 작성되어 있음.)

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
```

A* search 구현을 완료했다면 다음 명령어를 통해서 autograder testcase들을 모두 통과하는 지 확인하시오.

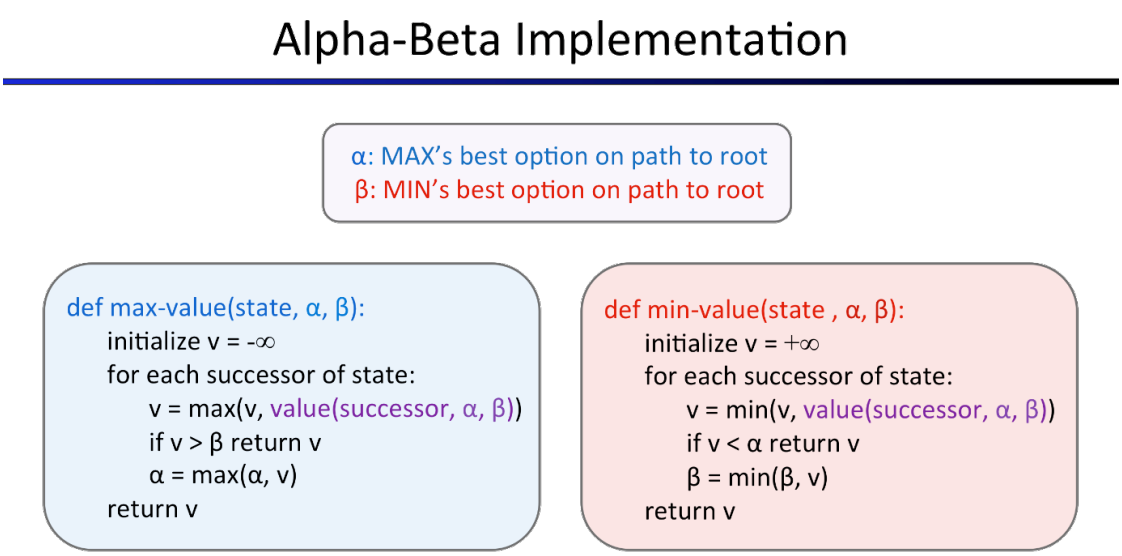
```
python autograder.py -q q4
```

2. Alpha-Beta pruning (5 points)

Alpha-Beta Pruning 구현에 필요한 코드는 `multiagent/multiAgent.py`의 `AlphaBetaAgent` 클래스에 작성하시오. 코드 작성에는 다음과 같은 주의사항이 있다.

1. 자식 순서를 재정렬하지 않고 alpha-beta pruning을 구현해야 한다. 즉,
   A. successor 상태는 `GameState.getLegalActions`에 의해 반환된 순서대로 처리되어야 한다.
   B. `GameState.generateSuccess`를 필요 이상으로 호출하지 말아야 한다.
2. autograder에 의해 탐색된 상태 집합들과 일치시키기 위해 가지치기를 시도하면 안 된다.

이 알고리즘의 pseudo-code는 다음과 같다.



Alpha-Beta Pruning 구현은 다음 명령어를 통해서 테스트 및 디버깅을 할 수 있다.

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

올바르게 구현했어도, 위 명령어를 실행하면 테스트에서 게임을 지는 경우가 발생할 수 있다. 이는 문제가 아니지만 autograder의 모든 질문들은 통과해야 한다.

3. Q-learning (6 points)

이번 구현 문제에서는 Q-learning을 이용해서 agent를 학습시킨다. 다음은 구현해야 하는 문제의 요약이다.

1. GridWorld 환경에서 Q-learning을 구현한다.
2. 그 후, Pac-Man 환경에 대해서 Q-learning을 적용 및 Approximate Q-learning을 구현한다.

`Q-Learning` 구현에 필요한 코드는 `reinforcement/qlearningAgents.py`에 작성한다. 먼저, `QLearningAgent` 클래스의 `update`와 `computeValueFromQValue`, `getQValue`, `computeActionFromQValues` 메소드들을 구현한다.

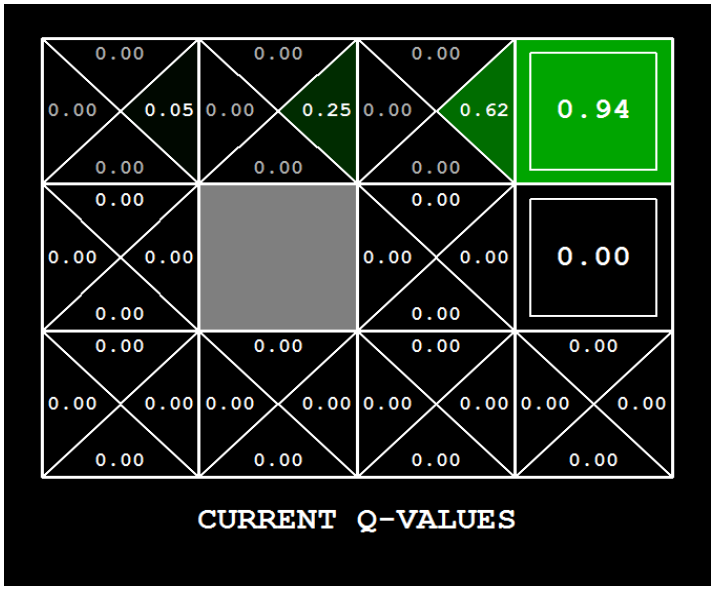
코드 작성시의 주의사항은 다음과 같다.

1. `computeActionFromQValues` 메소드의 경우, 더 좋은 동작을 위해 랜덤하게 action을 선택해야 하며 `random.choice()` 함수가 도움이 될 것이다.
2. Agent가 이전에 보지 못한 특정 상태와 action에 대한 Q-value는 0이어야 한다. 만약 agent가 경험한 모든 action들에 대한 Q-value가 음수라면 보지 못한 action이 optimal일 수도 있다.
3. `computeValueFromQValues`와 `computeActionFromQValues` 메소드에서 Q value를 접근할 때 오직 `getQValue` 메소드만을 사용할 수 있다.

Q-learning에 필요한 코드 구현을 완료한 경우, 다음 명령어를 통해서 키보드를 입력함에 따라 Q-Value가 최신화되는 것을 볼 수 있을 것이다.

```
python gridworld.py -a q -k 5 -m
```

`-k`는 agent가 학습할 수 있는 episode 개수를 제어하는 옵션이다. (Hint: noise를 제거하면 디버깅에 도움이 될 수 있으며, noise 제거는 `--noise 0.0` 옵션을 사용하면 된다.) 올바르게 코드를 작성했다면, 수동적으로 Pacman을 optimal 경로에 따라 북쪽으로 이동한 다음 동쪽으로 이동하는 것을 4 episode 동안 반복한 결과로 아래 그림과 같은 Q-value가 표시되어야 한다.



Q-learning을 올바르게 구현했다면 아래 명령어를 실행해서 autograder의 모든 질문을 통과해야만 한다.

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

4. Epsilon Greedy (2 points)

`QLearningAgent` 클래스의 `getAction` 메소드를 epsilon-greedy 방법을 적용해서 구현하시오. 구현 시 주의사항은 임의로 선택한 action은 best action일 수도 있다는 것이다. 즉, best action을 제외한 action 중 임의로 action이 선택되는 것이 아니라, action들 중 임의로 하나가 선택되는 것이다.

`random.choice` 함수를 호출하면 리스트에서 원소를 균일하게 임의로 선택할 수 있다. p의 확률로 True를 반환하고 1-p의 확률로 False를 반환하는 `util.flipCoin(p)`를 사용하면 성공할 확률이 p인 이진 변수를 시뮬레이션 할 수 있다.

`getAction` 메소드를 구현 완료했다면, 다음 명령어를 실행해서 GridWorld에서의 agent가 기대한 대로 행동하는지 관찰해 보시오.

```
python gridworld.py -a q -k 100
```

아래와 같이 다양한 epsilon 값들에 대해서도 시뮬레이션할 수 있다.

```
python gridworld.py -a q -k 100 --noise 0.0 -e 0.1
```

```
python gridworld.py -a q -k 100 --noise 0.0 -e 0.9
```

epsilon-greedy 구현에 대한 테스트는 다음과 같이 수행한다.

```
python autograder.py -q q4
```

또한 추가 코드 없이, Q-learning crawler robot을 실행할 수 있어야 한다.

```
python crawler.py
```

지금까지 작성된 코드는 GridWorld에 최적화 되어 있다면, 위의 코드가 동작하지 않기 때문에 모든 MDPs에 적용할 수 있도록 수정해야 한다.

5. Q-Learning and Pacman (2 points)

이제 PacMan 게임에 Q-Learning을 적용할 것이다. Q-Learning으로 PacMan 게임 agent를 훈련하는 것은 두 단계로 나눠진다. 첫 번째 단계는 학습으로, PacMan Agent는 position과 action의 가치에 대해서 배우기 시작할 것이다. 다만, 작은 grid 환경에서 Q-value를 학습하는 것 또한 매우 오랜 시간이 걸리기 때문에 GUI(또는 console) 화면 없이 학습이 수행될 것이다. 학습이 완료되면 바로 테스트 모드로 변경되어서 Agent가 게임을 수행하게 된다. 테스트 모드는 학습된 정책을 활용할 수 있도록 self.epsilon 및 self.alpha를 0으로 설정함에 따라 학습과 탐색을 비활성 시킨다. 테스트 게임은 기본적으로 GUI에 표시된다. 이전에 작성한 코드의 어떠한 수정도 없이 다음 명령어를 통해서 `smallGrid` Pacman에 Q-learning을 적용할 수 있어야 한다.

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

`PacmanQAgent`는 이전에 작성한 `QLearningAgent`을 상속받도록 설계되어 있지만 모든 기능이 동일하다. 유일하게 다른 점은 Pacman 문제에 더 효과적인 학습 파라미터(epsilon=0.05, alpha=0.2, gamma=0.8)를 default로 가지고 있다는 점이다. 위의 명령어가 아무런 예외 없이 작동하고 테스트 모드에서 80% 이상의 승률을 획득한다면 이 문제는 만점을 받을 수 있다.

(힌트) `QLearningAgent`가 `gridworld.py`와 `crawler.py`에 대해서 작동하지만 smallGrid Pacman에 대해서는 좋은 정책을 학습하지 못하는 것으로 보인다면, 이 문제는 `computeActionFromQValues` 메소드가 보지 못한 action들을 제대로 고려하지 않기 때문일 수도 있다. 정의상 보지 못한 action에 대한 Q-value는 0이기 때문에 경험한 모든 action에 대한 Q-value가 음수라면 보지 못한 action이 optimal일수도 있다. 필요시 `util.Counter`에서 argmax 함수를 이용할 수 있다.

작성한 코드는 아래 명령어를 통해서 채점할 수 있다.

```
python autograder.py -q q5
```

구현에 필요한 참고 사항은 다음과 같다.

1. 만약 학습 파라미터를 변경해서 실험을 하고 싶다면, `-a` 옵션을 사용하면 된다. 예를 들면 `-a epsilon=0.1,alpha=0.3,gamma=0.7`와 같이 사용할 수 있다.
2. `–n` 인자는 총 게임 개수를 가리키고 `–x`는 학습에 사용하고자 하는 게임 개수를 가리킨다. 즉, `–x 2000 –n 2010` 옵션은 총 2010개의 게임 중 처음 2000개는 학습을 진행하고 나머지 10개의 게임을 테스트 하겠다는 것을 의미한다.
3. 10개의 게임 학습을 진행할 때 동시에 진행상황을 확인하고 싶다면 다음의 명령어를 사용하면 된다.

   ```
   python pacman.py -p PacmanQAgent -n 10 -l smallGrid -a numTraining=10
   ```

4. 훈련 중에는 100 게임마다 학습 진행 상황이 출력된다. 좋은 정책을 학습한 Pacman agent라도 epsilon이 양수이면 임의로 action을 선택하기 때문에 가끔 유령이 있는 위치로 이동할 것이다.
5. 지난 100 episode 동안 Pacman의 보상이 양수가 되기까지 1000에서 1400 게임이 걸릴 것이며, 이는 Pacman이 이기기를 시작했다는 것을 의미한다. 학습이 완료될 무렵, 보상 값은 여전히 양수이면서 상당히 높은 값(100 ~ 350)이어야 한다.
6. 학습을 마친 Pacman은 테스트 모드에서 매우 안정적으로 승리해야 한다. (적어도 90% 이상)

smallGrid 환경에서 Pacman을 잘 학습시킬 수 있었지만 mediumGrid 환경에서는 학습이 동작하지 않는다. 평균

훈련 보상은 훈련 내내 음수를 유지하고 테스트를 진행 시 모든 게임에서 패배할 것이며 학습에도 오랜 시간이 걸리는 문제가 있다.

현재 Q-Learn은 보드 configuration이 변경되면 그에 따라서 Q-Value들을 새로 학습해야 한다. 즉, 다양한 보드 크기를 가진 Pacman 게임 모두에 대해 적용할 수 있는 정책을 학습할 수 없다. 이는 현재 접근 방법으로 모든 위치에서 마주치는 유령이 나쁘다는 것을 일반화할 수 없다는 것을 의미한다.

6. Approximate Q-Learning (4 points)

상태의 feature에 대한 weight를 학습하는 approximate Q-learning agent를 구현하라. approximate Q-learning 구현에 필요한 코드는 `reinforcement/qlearningAgents.py`의 `PacmanQAgent`를 상속받은 `ApproximateQAgent` 클래스에 작성하시오.

Approximate Q-learning은 상태 s와 action a 쌍을 입력 받아 $[\mathrm{f}_1(\mathrm{s},\mathrm{a}),\ldots \mathrm{f}_\mathrm{n}(\mathrm{s},\mathrm{a})]$를 출력하는 $\mathrm{f}(\mathrm{s},\mathrm{a})$가 존재한다고 가정한다. feature를 직접 설계해서 구현할 수도 있지만 이 문제에서는 그럴 필요 없이 `featureExtrators.py`에서 제공된다. 제공되는 feature vector는 0이 아닌 feature와 값의 쌍으로 이루어진 `util.Counter` (dictionary와 비슷함) 객체이며, 이때 경험하지 못한 상태와 action에 대한 feature의 값은 0이다. 본 과제에서는 vector의 index가 어떤 feature를 나타내는 대신에 dictionary의 key로 feature의 identity를 정의하였다.

Approximate Q-function은 다음과 같은 형태를 지닌다.

$$\mathrm{Q}(\mathrm{s},\mathrm{a}) = \sum_{i=1}^{n} f_i\,(s,a) w_i$$

코드 구현에서 가중치 벡터는 feature를 key로 weight를 값으로 갖는 dictionary로 구현해야 한다. 이전에 Q-values를 업데이트한 방법과 유사하게 가중치 벡터를 아래와 같이 업데이트할 것이다.

$$\mathrm{w_i} = \mathrm{w_i} + \alpha \cdot \mathrm{differnce} \cdot f_i(s,a)$$

$$\mathrm{differnce} = \left(r + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)$$

위 식에서 difference 항은 일반적인 Q-learning과 동일하며 r은 보상이다.

`ApproximateQAgent`는 하나의 feature를 상태와 action 쌍에 할당하는 `IdentityExtractor`를 사용하도록 default로 설정되어 있다. 이 feature extractor를 사용한 approximate Q-learning agent는 `PacmanQAgent`와 동일하게 작동해야 한다. 다음의 명령어를 통해서 `IdentityExtractor`를 사용한 approximate Q-learning agent를 테스트한다..

```
python pacman.py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
```

`ApproximateQAgent`는 `QLearningAgent`의 하위 클래스이므로 getAction과 같은 여러 메소드들을 공유한다. `ApproximateQAgent`에서 Q-value에 직접 접근하는 대신 `QLearningAgent`의 `getQValue`를 호출해서 접근하고 `getQValue`를 override 해서 새로운 approximate q-value가 action을 계산하는 데 이용한다.

Identity feature에 대한 approximate learner가 동작한다고 판단되면, 쉽게 이기는 법을 배울 수 있는 custom feature extractor에 대해서 approximate Q-learning을 실행한다. 명령어는 다음과 같다.

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid
```

올바르게 구현한 `ApproximateQAgent`는 `mediumGrid` 보다 더 큰 환경에서도 문제 없이 동작해야 한다.
(학습에 몇 분 정도의 시간이 걸릴 수 있음)

```
python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic
```

오류가 없는 경우 approximate Q-learning agent는 단 50번의 게임을 학습하더라도 거의 매번 승리한다.

구현한 approximate Q-learning agent는 reference implementation의 Q-Value와 feature weight와 함께 비교해서 같은 값을 갖는 지 확인할 것이다. 작성한 코드는 아래 명령어를 통해서 autograder로 채점할 수 있다.

```
python autograder.py -q q6
```