Assignment2 - Pacman Game

MinimaxAgent, AlphaBetaAgent, ExpectimaxAgent - 20121277 김주호

알고리즘 구현 방법

MinimaxAgent 클래스의 Action 메서드

- 3 가지 함수를 선언하였다.
  - `def minimax_handler(state, agent_idx, depth)`
  - `def max_player(state, agent_idx, depth)`
  - `def min_player(state, agent_idx, depth)`
- max player 는 nxt state 들이 만든 return 값 중 최대값을 return 하며, min_player 함수를 재귀적으로 call 한다.
- min player 는 nxt state 들이 만든 return 값 중 최소값을 return 한다.
  - 현재 agent 에 따라 다른 재귀함수를 call 하게 된다.
  - next agent 가 ghost 인 경우, 또 다른 ghost 의 min player 를 call 한다.
  - next agent 가 pacman 인 경우, max player 를 call 하며, depth 를 하나 증가시킨다.
- depth 가 `self.depth` 가 되는 경우 말단 노드에 도달한 것이고, self.evaluationFunction(state)를 돌려주게 된다.
- `minimax_handler()`는 이 모든 함수 call 의 흐름을 제어한다.

AlphaBetaAgent 클래스의 Action 메서드

- 3 가지 함수를 선언하였다.
  - `def alphabeta_handler(state, depth, agent_index, alpha=-INF, beta=INF)`
  - `def max_player(state, depth, agent_index, alpha=-INF, beta=INF)`
  - `def min_player(state, depth, agent_index, alpha=-INF, beta=INF)`
- 기본적인 함수 call 의 흐름은 minimax agent 와 동일하며, 이를 `alphabeta_handler()`가 제어한다.
- minimax agent 와 다른 점은 함수에 alpha, beta 인자를 추가하여, alpha, beta 값과 노드의 값을 비교하여 pruning 의 개념을 추가한 점이다.

```
## max player 의 경우
mxv = max(mxv, eval)
alpha = max(alpha, eval)
if alpha >= beta:  # pruning
```

```
            return mxv
    .
    .
    .
    ## min player 의 경우
    mnv = min(mnv, eval)
    beta = min(beta, eval)
    if beta <= alpha:  # pruning
            return mnv
```

- alpha 값은 - inf 로 beta 값은 inf 로 초기화한다.
- max player 의 경우, 모든 successor state 들을 보면서 `self.evaluationFunction(state)`을 통해 넘겨받은 리턴 값과 alpha 값의 maximum 값으로 alpha 값을 갱신한다.
- 이 갱신한 alpha 값이 beta 값 이상의 값을 갖는 경우, pruning 을 수행한다.
- 왜냐하면, 그 위의 min_player 가 beta 값 이하의 값을 취하는 것이 개런티 되어있는데, beta 이상의 값을 넘겨 받는 순간 해당 가지는 min_player 의 선택을 받을 수 없는 가지이므로 잘라내도 되는 것이다.
- 마찬가지로, min player 도 successor 가 alpha 보다 작거나 같은 beta 값으로 갱신이 될때, 가지치기로 잘라낼 수 있다.
- 왜냐하면, 그 위의 max_player 가 alpha 값 이상의 값을 취하는 것이 개런티 되어있는데, 그 alpha 값보다도 작은 값을 return 한다면, 이는 max_player 의 선택을 받을 수 없는 가지이므로 잘라낼 수 있다.

ExpectimaxAgent 클래스의 Action 메서드

- 3 가지 함수를 선언하였다.
  - `def expectimax_handler(state, agent, depth)`
  - `def max_player(state, agent, depth)`
  - `def exp_player(state, agent, depth)`
- 기본적인 함수 call 의 흐름은 minimax agent 와 동일하며, 이를 `expectimax_handler()`가 제어한다.

- min player 의 동작 대신 exp player 의 동작으로 대신하면 된다.
- exp player 는 min player 와는 다르게 nxt state 값의 최소를 선택하는 것이 아니라, 각 state 를 선택할 가능성이 확률적으로 분배되어있다고 본다.
- 해당 agent 의 action 에서는 그 확률이 **uniformly distributed 되어있다는 가정**을 하고, 균등하게 확률을 분배하였다.

```
p = 1 / len(state.getLegalActions(agent))
```

- exp player 는 위에서 정의한 확률대로, successor 들에게서 기대되는 값을 return 하게 된다.

```
exp_val = 0
for action in state.getLegalActions(agent):
      nxt_state = state.generateSuccessor(agent, action)
      p = 1 / len(state.getLegalActions(agent))
      exp_val += p * expectimax_handler(nxt_state, nxt_agent, depth)
      return exp_val
```

Agent 실행

Minimax Agent

다음의 명령어로, minimaxmap 에서 minimax agent 의 승률을 확인한다.

```
python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q
```

실험 1 결과

```
❯ python pacman.py -p MinimaxAgent -m minimaxmap -a depth=4 -n 1000 -q
====================================
[1] Pacman Lose... Score: -492
[2] Pacman Lose... Score: -492
[3] Pacman Lose... Score: -492
[4] Pacman Lose... Score: -495
[5] Pacman Lose... Score: -492
[6] Pacman Lose... Score: -492
[7] Pacman Win! Score: 516
[8] Pacman Win! Score: 516
[9] Pacman Win! Score: 516
[10] Pacman Win! Score: 516
[11] Pacman Win! Score: 514
[12] Pacman Lose... Score: -495
[13] Pacman Lose... Score: -492
[14] Pacman Win! Score: 516
[15] Pacman Win! Score: 516
```

```
[991] Pacman Lose... Score: -492
[992] Pacman Win! Score: 516
[993] Pacman Win! Score: 516
[994] Pacman Win! Score: 516
[995] Pacman Lose... Score: -494
[996] Pacman Win! Score: 514
[997] Pacman Lose... Score: -492
[998] Pacman Win! Score: 516
[999] Pacman Win! Score: 516
[1000] Pacman Lose... Score: -492
----------Game Results----------
```

```
Win, Lose, Win, Lose, Win, Win, Lose

Win Rate: 63% (637/1000)
Total Time: 91.63857936859131
Average Time: 0.09163857936859131
------------------------------------
```

실험 2 결과

```
Win, Win, Lose, Win, Lose, Win, Win, Lose

Win Rate: 63% (633/1000)
Total Time: 94.90069961547852
Average Time: 0.09490069961547852
--------------------------------------
```

실험 3 결과

```
Win, Lose, Lose, Win, Win, Lose, Win

Win Rate: 63% (638/1000)
Total Time: 97.34019923210144
Average Time: 0.09734019923210144
======================================
```

=> minimaxmap 에서 minimax agent 의 승률이 50% ~ 70%가 됨을 확인할 수 있다.

AlphaBeta Pruning Agent

다음의 명령어로 minimax agent 와 alphabeta agent 의 수행시간을 비교하여, alphabeta agent 의 pruning 이 잘 작동하는지 확인한다.

```
python time_check.py
```

실험 결과

- minimax, depth=2, smallmap

```
Win Rate: 0% (2/300)
Total Time: 48.28879690170288
Average Time: 0.1609626563390096
======================================
----- END  MiniMax (depth=2) For Small Map
```

- alphabeta, depth=2, smallmap

```
Win Rate: 0% (1/300)
Total Time: 40.824827671051025
Average Time: 0.1360827589035034
====================================
----- END  AlphaBeta (depth=2) For Small Map
```

- minimax, depth=2, mediummap

```
Win Rate: 4% (14/300)
Total Time: 121.16286492347717
Average Time: 0.40387621641159055
====================================
----- END  MiniMax (depth=2) For Medium Map
```

- alphabeta, depth=2, mediummap

```
Win Rate: 2% (7/300)
Total Time: 99.17087006568909
Average Time: 0.3305695668856303
====================================
----- END  AlphaBeta (depth=2) For Medium Map
```

- minimax, depth=4, minimaxmap

```
Win Rate: 36% (365/1000)
Total Time: 75.23743462562561
Average Time: 0.07523743462562561
====================================
----- END  MiniMax (depth=4) For Minimax Map
```

- alphabeta, depth=4, minimaxmap

```
Win Rate: 39% (393/1000)
Total Time: 41.55166959762573
Average Time: 0.04155166959762573
====================================
----- END  AlphaBeta (depth=4) For Minimax Map
```

정리

	minimax	alphabeta
iteration 300, smallmap, depth=2	Tot = 48.28Avg = 0.16	Tot = 40.82Avg = 0.13
iteration 300, mediummap, depth=2	Tot = 121.16Avg = 0.403	Tot = 99.17Avg = 0.33
iteration 1000, minimaxmap, depth=4	Tot = 75.23Avg = 0.07	Tot = 41.55Avg = 0.04

=> alphabeta agent 가 minimax agent 보다 효율적임을 알 수 있다. depth 를 깊게 실험해보면, alphabeta agent 의 pruning 효과가 더욱 극명하다.

Expectimax Agent

- 다음의 명령어로 stuckmap 에서 expectimax agent 가 50%의 승률을 얻는지 확인한다.
- 이긴 경우에 `532 score`, 진 경우에 `- 502 score` 를 얻는지 관찰하여 expectimax agent 의 동작을 점검한다.

```
python pacman.py -p ExpectimaxAgent -m stuckmap -a depth=3 -n 100 -q
```

실험 1 결과

```
❯ python pacman.py -p ExpectimaxAgent -m stuckmap -a depth=3 -n 100 -q
====================================
[1] Pacman Lose... Score: -502
[2] Pacman Lose... Score: -502
[3] Pacman Win! Score: 532
[4] Pacman Lose... Score: -502
[5] Pacman Win! Score: 532
[6] Pacman Lose... Score: -502
[7] Pacman Win! Score: 532
[8] Pacman Win! Score: 532
[9] Pacman Lose... Score: -502
[10] Pacman Lose... Score: -502
[11] Pacman Win! Score: 532
```

```
Win Rate: 48% (48/100)
Total Time: 0.45209407806396484
Average Time: 0.004520940780639649
======================================
```

실험 2 결과

```
Win Rate: 52% (52/100)
Total Time: 0.47816920280456543
Average Time: 0.004781692028045654
======================================
```

실험 3 결과

```
Win Rate: 46% (46/100)
Total Time: 0.4535026550292969
Average Time: 0.004535026550292969
--------------------------------------
```

- stuckmap 에서 expectimax agent 가 50% 정도의 승률을 얻는다.
- 이긴 경우에 `532 score`, 진 경우에 `- 502 score` 를 얻는다.