AI Assignment02 보고서

```
AI Assignment02 보고서
학번 : 20171612
학과 : 컴퓨터공학과
이름 : 김성일
```

Minimax Agent

코드

```
class MinimaxAgent(AdversialSearchAgent):
  """
    [문제 01] MiniMax의 Action을 구현하시오. (20점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def isTerminalState(self, state, depth):
    return state.isWin() or state.isLose() or depth == self.depth

  def minimax(self, state, depth, agent, maximize):
    # Check terminal state
    if self.isTerminalState(state, depth):
      return {
        "action": None,
        "score": self.evaluationFunction(state)
      }

    # Init result
    result = {
      "action" : None,
      "score" : float("inf") if agent else float("-inf"),
    }

    # Iterate children of node
    for action in state.getLegalActions(agent):
      # Generate new state
      new_state = state.generateSuccessor(agent, action)
      # Maximizing agent
      if maximize:
        score = self.minimax(new_state, depth, 1, False)["score"]
        result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
      # Minimizing agent
      else:
        if agent >= state.getNumAgents() - 1:
          score = self.minimax(new_state, depth + 1, 0, True)["score"]
        else:
          score = self.minimax(new_state, depth, agent + 1, False)["score"]
        result["score"] = min(score, result["score"])

    return result

  def Action(self, gameState):
    # ######################## Write Your Code Here ################################
    # Call minimax
    return self.minimax(gameState, 0 ,0, True)["action"]
    ##############################################################################
```

- `isTerminalState`
  - 현재 state와 depth를 체크하여 terminal state인지 체크하여 True 혹은 False를 반환하는 함수입니다.
- `minimax`
  - mini-max 알고리즘을 이용하여 Agent의 Action을 반환하는 함수입니다.
  - state, depth, agent, maximize 라는 parameter를 전달받습니다.
    - state: 말그대로 game state를 전달받습니다.
    - depth: 현재 depth를 전달받습니다.

    - agent: 현재 agent를 전달받습니다. (0이라면 pacman 그게 아니라면, ghost입니다.)
    - maximize: 현재 agent가 maximize하는 agent인지 minimize하는 agent인지 나타내는 Boolean 값입니다.
  - terminal state인지 체크하고 terminal state라면 score를 반환합니다.

```
# Check terminal state
if self.isTerminalState(state, depth):
  return {
    "action": None,
    "score": self.evaluationFunction(state)
  }
```

  - `result`

    아래와 같이 action과 score를 함께 return하기위한 dictionary입니다.

    action은 None으로 초기화되며, score는 agent가 Pacman(=0)이라면 음의 무한대로 초기화되고 Ghost(≠0)라면 양의 무한대로 초기화됩니다.

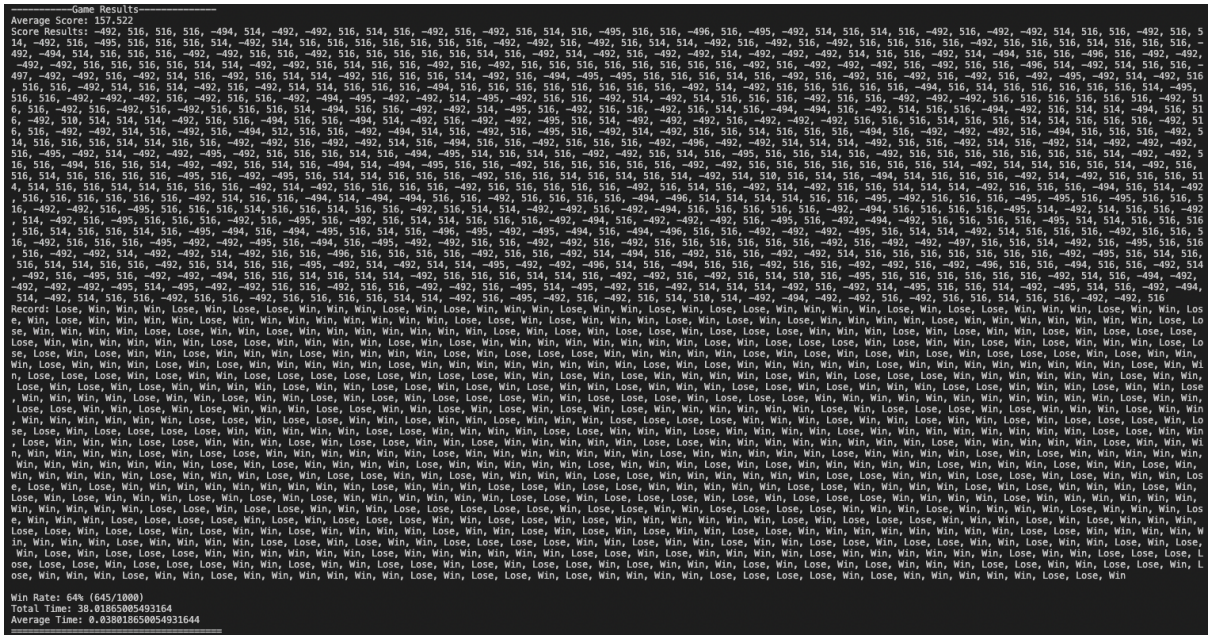
```
result = {
  "action" : None,
  "score" : float("inf") if agent else float("-inf"),
}
```

  - **동작과정**
    - 현재 agent를 기반으로 `state.getLegalActions` 를 통해 다음 action들을 iterate하면서 action과 score를 업데이트합니다.
    - 새로운 state는 현재 agent와 각 action을 이용하여 generateSuccessor를 통해 생성됩니다.
    - 각 depth에서 모든 agent(pacman 및 ghost)가 움직여야 depth가 증가하게 됩니다.
    - maximize를 통해 어떤 행동을 취하게 되는지 정해집니다.
      - 만약 maximize가 True라면 Pacman이기 때문에 minimax를 통해 depth를 유지하며 모든 agent가 움직일 수 있도록합니다.
        - 만약 새로 구한 score가 result score보다 크다면 result action과 result score를 update합니다.
      - 만약 maximize가 False라면 Ghost이기 때문에 마지막 Ghost까지 체크를 하였다면 (if agent >= state.getNumAgents() - 1), depth를 증가시키고 agent가 Pacman이 되도록합니다. 그렇지 않다면, depth를 유지하고 다음 agent가 움직이도록합니다.
        - 만약 새로구한 score가 result score보다 작다면 result score를 update합니다.
    - 이렇게 얻은 result dictionary를 반환합니다.

```
for action in state.getLegalActions(agent):
  # Generate new state
  new_state = state.generateSuccessor(agent, action)
  # Maximizing agent
  if maximize:
    score = self.minimax(new_state, depth, 1, False)["score"]
    result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
  # Minimizing agent
  else:
    if agent >= state.getNumAgents() - 1:
      score = self.minimax(new_state, depth + 1, 0, True)["score"]
    else:
      score = self.minimax(new_state, depth, agent + 1, False)["score"]
      result["score"] = min(score, result["score"])
return result
```

- `Action`
  - minimax 함수를 호출하여 `result["action"]` 을 얻고 그것을 return하는 함수입니다.

결과



승률은 64%이며 50% ~ 70% 사이의 승률임을 만족합니다.

Alpha Beta Agent

코드

```
class AlphaBetaAgent(AdversialSearchAgent):
  """
    [문제 02] AlphaBeta의 Action을 구현하시오. (25점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def isTerminalState(self, state, depth):
    return state.isWin() or state.isLose() or depth == self.depth

  def alpha_beta(self, state, depth, agent, maximize, alpha, beta):
    # Check terminal state
    if self.isTerminalState(state, depth):
      return {
        "action": None,
        "score": self.evaluationFunction(state)
      }

    # Init result
    result = {
      "action" : None,
      "score" : float("inf") if agent else float("-inf"),
    }

    # Iterate children of node
    for action in state.getLegalActions(agent):
      # Generate new state
      new_state = state.generateSuccessor(agent, action)
      # Maximizing agent
      if maximize:
        score = self.alpha_beta(new_state, depth, 1, False, alpha, beta)["score"]
        result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
        # Pruning
        if result["score"] >= beta:
          break
        alpha = max(alpha, result["score"])
      # Minimizing agent
      else:
```

```
        if agent == state.getNumAgents() - 1:
          score = self.alpha_beta(new_state, depth + 1, 0, True, alpha, beta)["score"]
        else:
          score = self.alpha_beta(new_state, depth, agent + 1, False, alpha, beta)["score"]
        result["score"] = min(score, result["score"])
        # Pruning
        if alpha >= result["score"]:
          break
        beta = min(beta, result["score"])
    return result

  def Action(self, gameState):
    ###################### Write Your Code Here ################################
    # Call alpha beta
    return self.alpha_beta(gameState, 0 ,0, True, float("-inf"), float("inf"))["action"]
    ############################################################################
```

위에서 구현한 Minimax 알고리즘에 alpha-beta pruning 방법을 적용하여 구현하였습니다.

- `isTerminalState`
  - 현재 state와 depth를 체크하여 terminal state인지 체크하여 True 혹은 False를 반환하는 함수입니다.
- `alpha_beta`
  - minimax Alpha Beta Pruing 알고리즘을 이용하여 Agent의 Action을 반환하는 함수입니다.
  - state, depth, agent, maximize, alpha, beta 라는 parameter를 전달받습니다.
    - state: 말그대로 game state를 전달받습니다.
    - depth: 현재 depth를 전달받습니다.
    - agent: 현재 agent를 전달받습니다. (0이라면 pacman 그게 아니라면, ghost입니다.)
    - maximize: 현재 agent가 maximize하는 agent인지 minimize하는 agent인지 나타내는 Boolean 값입니다.
    - alpha: Alpha Beta Pruing에서 쓰이는 Alpha 값입니다. 처음에는 음의 무한대를 전달 받습니다.
    - beta: Alpha Beta Pruing에서 쓰이는 Beta 값입니다. 처음에는 양의 무한대를 전달 받습니다.
  - terminal state인지 체크하고 terminal state라면 score를 반환합니다.

    ```
    # Check terminal state
    if self.isTerminalState(state, depth):
      return {
        "action": None,
        "score": self.evaluationFunction(state)
      }
    ```

  - `result`

    아래와 같이 action과 score를 함께 return하기위한 dictionary입니다.

    action은 None으로 초기화되며, score는 agent가 Pacman(=0)이라면 음의 무한대로 초기화되고 Ghost(≠0)라면 양의 무한대로 초기화됩니다.

    ```
    result = {
      "action" : None,
      "score" : float("inf") if agent else float("-inf"),
    }
    ```

  - **동작과정**
    - 현재 agent를 기반으로 `state.getLegalActions` 를 통해 다음 action들을 iterate하면서 action과 score를 업데이트합니다.
    - 새로운 state는 현재 agent와 각 action을 이용하여 generateSuccessor를 통해 생성됩니다.
    - 각 depth에서 모든 agent(pacman 및 ghost)가 움직여야 depth가 증가하게 됩니다.
    - maximize를 통해 어떤 행동을 취하게 되는지 정해집니다.

- 만약 maximize가 True라면 Pacman이기 때문에 minimax를 통해 depth를 유지하며 모든 agent가 움직일 수 있도록합니다.
    - 만약 새로 구한 score가 result score보다 크다면 result action과 result score를 update합니다.
    - 만약 beta 값이 result score보다 작거나 같으면 **pruning을 진행하여 break합니다.**
    - result score가 alpha 값보다 크다면 alpha 값을 update합니다.
- 만약 maximize가 False라면 Ghost이기 때문에 마지막 Ghost까지 체크를 하였다면 (if agent >= state.getNumAgents() - 1), depth를 증가시키고 agent가 Pacman이 되도록합니다. 그렇지 않다면, depth를 유지하고 다음 agent가 움직이도록합니다.
    - 만약 새로구한 score가 result score보다 작다면 result score를 update합니다.
    - 만약 alpha 값이 result score보다 크거나 같으면 **pruning을 진행하여 break합니다.**
    - result scor가 beta 값보다 작다면 beta 값을 update합니다.

▪ 이렇게 얻은 result dictionary를 반환합니다.

```
# Iterate children of node
for action in state.getLegalActions(agent):
  # Generate new state
  new_state = state.generateSuccessor(agent, action)
  # Maximizing agent
  if maximize:
    score = self.alpha_beta(new_state, depth, 1, False, alpha, beta)["score"]
    result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
    # Pruning
    if result["score"] >= beta:
      break
    alpha = max(alpha, result["score"])
  # Minimizing agent
  else:
    if agent == state.getNumAgents() - 1:
      score = self.alpha_beta(new_state, depth + 1, 0, True, alpha, beta)["score"]
    else:
      score = self.alpha_beta(new_state, depth, agent + 1, False, alpha, beta)["score"]
      result["score"] = min(score, result["score"])
      # Pruning
      if alpha >= result["score"]:
        break
      beta = min(beta, result["score"])
return result
```

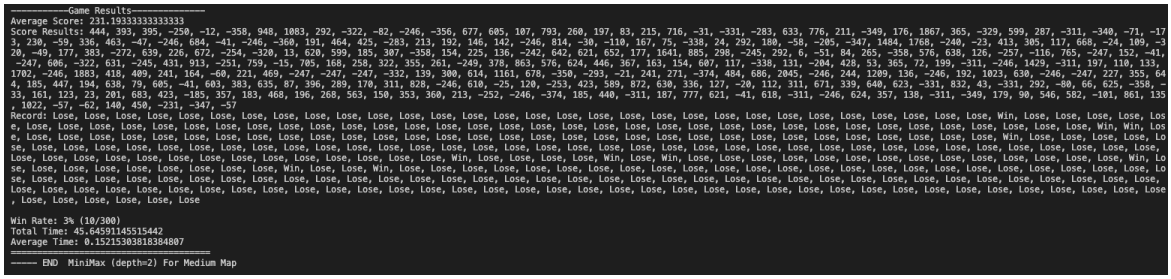
- `Action`
    - alpha_beta 함수를 호출하여 `result["action"]`을 얻고 그것을 return하는 함수입니다.

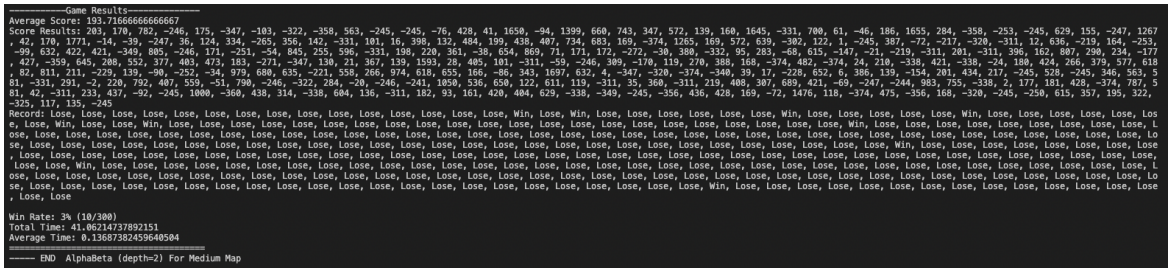
결과

- **Small Map**

- **MiniMax** Execute Time : 0.070350 sec
- **AlphaBeta** Execute Time : 0.064396 sec

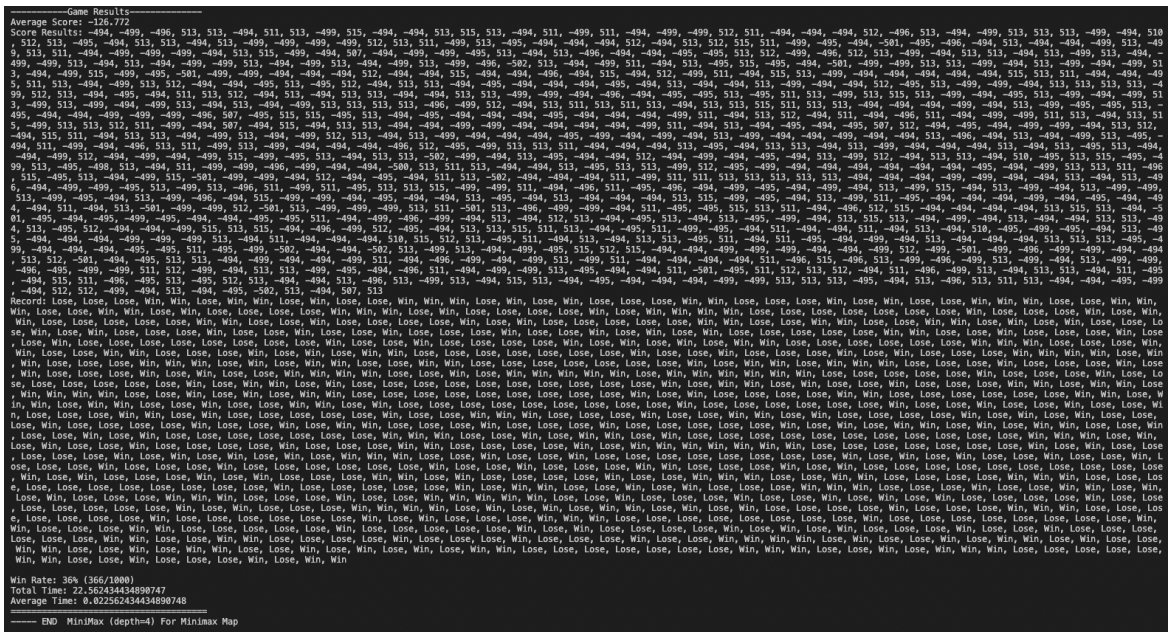
- **Medium Map**

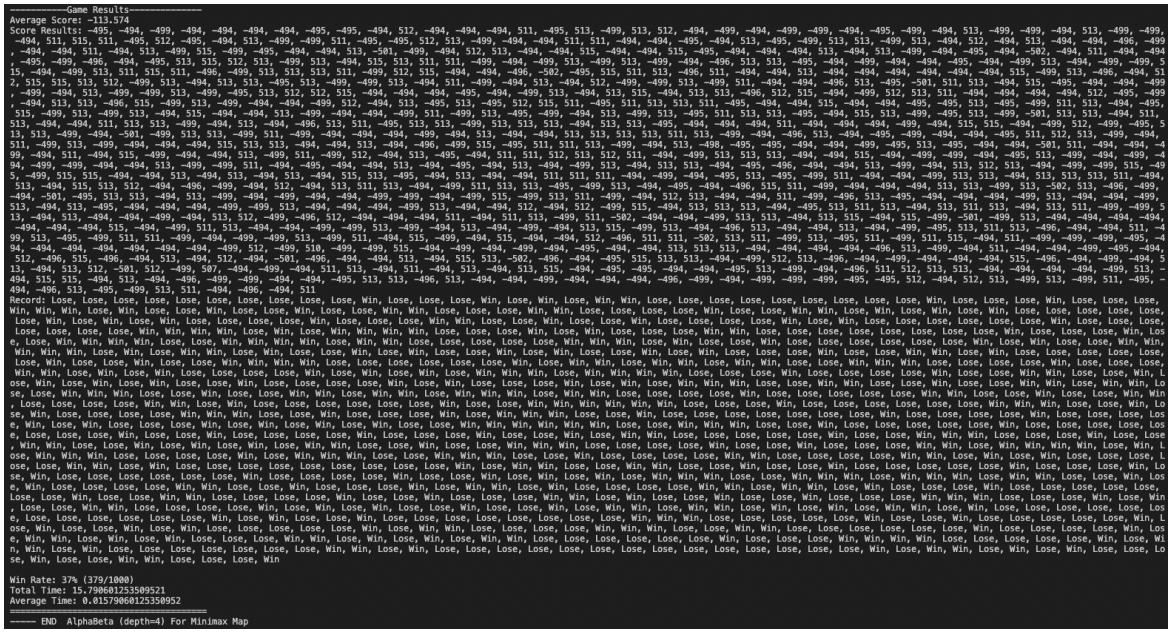




- **MiniMax** Average Execute Time : 0.1521530 sec
- **AlphaBeta** Average Execute Time : 0.136873sec

- **Minimax Map**





- **MiniMax** Average Execute Time : 0.0225624 sec
- **AlphaBeta** Average Execute Time : 0.0157906 sec

위와 같이 Alpha Beta Agent가 Minimax Agent 보다 빠르게 끝나는 결과를 보인다는 것을 알 수 있습니다.

Expectimax Agent

코드

```
class ExpectimaxAgent(AdversialSearchAgent):
  """
    [문제 03] Expectimax의 Action을 구현하시오. (25점)
    (depth와 evaluation function은 위에서 정의한 self.depth and self.evaluationFunction을 사용할 것.)
  """
  def isTerminalState(self, state, depth):
    return state.isWin() or state.isLose() or depth == self.depth

  def expectimax(self, state, depth, agent, maximize):
    # Check terminal state
    if self.isTerminalState(state, depth):
      return {
        "action": None,
        "score": self.evaluationFunction(state)
      }

    # Init result
    result = {
      "action" : None,
      "score" : 0.0 if agent else float("-inf"),
    }

    # Probability function
    prob = lambda x: x/len(state.getLegalActions(agent))

    # Iterate children of node
    for action in state.getLegalActions(agent):
      # Generate new state
      new_state = state.generateSuccessor(agent, action)
      # Maximizing agent
      if maximize:
        score = self.expectimax(new_state, depth, 1, False)["score"]
        result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
      # Minimizing agent
      else:
        if agent >= state.getNumAgents() - 1:
          score = self.expectimax(new_state, depth + 1, 0, True)["score"]
```

```
        else:
          score = self.expectimax(new_state, depth, agent + 1, False)["score"]
        result["score"] += prob(score)
    return result

  def Action(self, gameState):
    ###################### Write Your Code Here ################################
    # Call expectimax
    return self.expectimax(gameState, 0 ,0, True)["action"]
    ############################################################################
```

위에서 구현한 Minimax 알고리즘을 수정하여, Expactimax를 구현하였습니다.

- `isTerminalState`
  - 현재 state와 depth를 체크하여 terminal state인지 체크하여 True 혹은 False를 반환하는 함수입니다.
- `expectimax`
  - expectimax 알고리즘을 이용하여 Agent의 Action을 반환하는 함수입니다.
  - state, depth, agent, maximize 라는 parameter를 전달받습니다.
    - state: 말그대로 game state를 전달받습니다.
    - depth: 현재 depth를 전달받습니다.
    - agent: 현재 agent를 전달받습니다. (0이라면 pacman 그게 아니라면, ghost입니다.)
    - maximize: 현재 agent가 maximize하는 agent인지 minimize하는 agent인지 나타내는 Boolean 값입니다.
  - terminal state인지 체크하고 terminal state라면 score를 반환합니다.

    ```
    # Check terminal state
    if self.isTerminalState(state, depth):
      return {
        "action": None,
        "score": self.evaluationFunction(state)
      }
    ```

  - `result`

    아래와 같이 action과 score를 함께 return하기위한 dictionary입니다.

    action은 None으로 초기화되며, score는 agent가 Pacman(=0)이라면 음의 무한대로 초기화되고 Ghost(≠0)라면 0으로 초기화됩니다.

    ```
    result = {
      "action" : None,
      "score" : 0.0 if agent else float("-inf"),
    }
    ```

  - **동작과정**
    - 현재 agent를 기반으로 `state.getLegalActions` 를 통해 다음 action들을 iterate하면서 action과 score를 업데이트합니다.
    - 새로운 state는 현재 agent와 각 action을 이용하여 generateSuccessor를 통해 생성됩니다.
    - 각 depth에서 모든 agent(pacman 및 ghost)가 움직여야 depth가 증가하게 됩니다.
    - `prob`
      - Expactimax를 위해 확률 계산을 진행하는 함수입니다.
      - `x * (1/len(state.getLegalActions(agent)))` 을 return하는 함수입니다.
    - maximize를 통해 어떤 행동을 취하게 되는지 정해집니다.

        - 만약 maximize가 True라면 Pacman이기 때문에 minimax를 통해 depth를 유지하며 모든 agent가 움직일 수 있도록합니다.
            - 만약 새로 구한 score가 result score보다 크다면 result action과 result score를 update합니다.
        - 만약 maximize가 False라면 Ghost이기 때문에 마지막 Ghost까지 체크를 하였다면 (if agent >= state.getNumAgents() - 1), depth를 증가시키고 agent가 Pacman이 되도록합니다. 그렇지 않다면, depth를 유지하고 다음 agent가 움직이도록합니다.
            - prob함수에게 새로 구한 score를 전달하여 반환 받은 값을 result score에 더하여 update합니다.
    - 이렇게 얻은 result dictionary를 반환합니다.

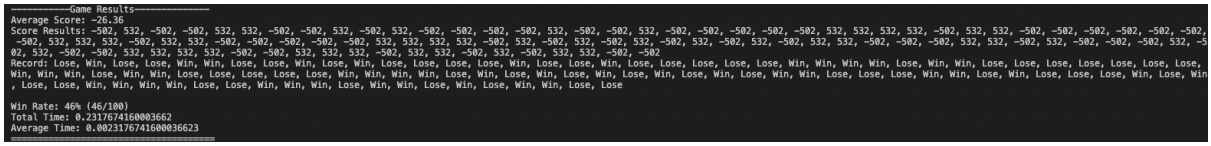
```
# Probability function
prob = lambda x: x/len(state.getLegalActions(agent))

# Iterate children of node
for action in state.getLegalActions(agent):
  # Generate new state
  new_state = state.generateSuccessor(agent, action)
  # Maximizing agent
  if maximize:
    score = self.expectimax(new_state, depth, 1, False)["score"]
    result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
  # Minimizing agent
  else:
    if agent >= state.getNumAgents() - 1:
      score = self.expectimax(new_state, depth + 1, 0, True)["score"]
    else:
      score = self.expectimax(new_state, depth, agent + 1, False)["score"]
    result["score"] += prob(score)
return result
```

- `Action`
    - expectimax 함수를 호출하여 `result["action"]`을 얻고 그것을 return하는 함수입니다.

결과

100번의 게임을 실행했을 때 Score와 Win rate



- Expectimax Agent를 사용하여 게임을 진행하면 Win인 경우 **532**의 Score를, Lose인 경우 **-502**의 Score를 보여줍니다.
- Expectimax Agent를 사용한 게임을 100번 진행하였을 때는 **Win rate가 46%가 나오며, 50% 정도**임을 알 수 있습니다.

승률의 분포를 통계적으로 확인

1~ 1000번의 게임을 실행했을 때 승률의 분포

이것을 더 확실하게 살펴보기 위해 1번부터 1000번까지 각각의 게임을 진행해보았습니다.

- **1~1000번의 게임을 진행하기 위한 쉘 스크립트**

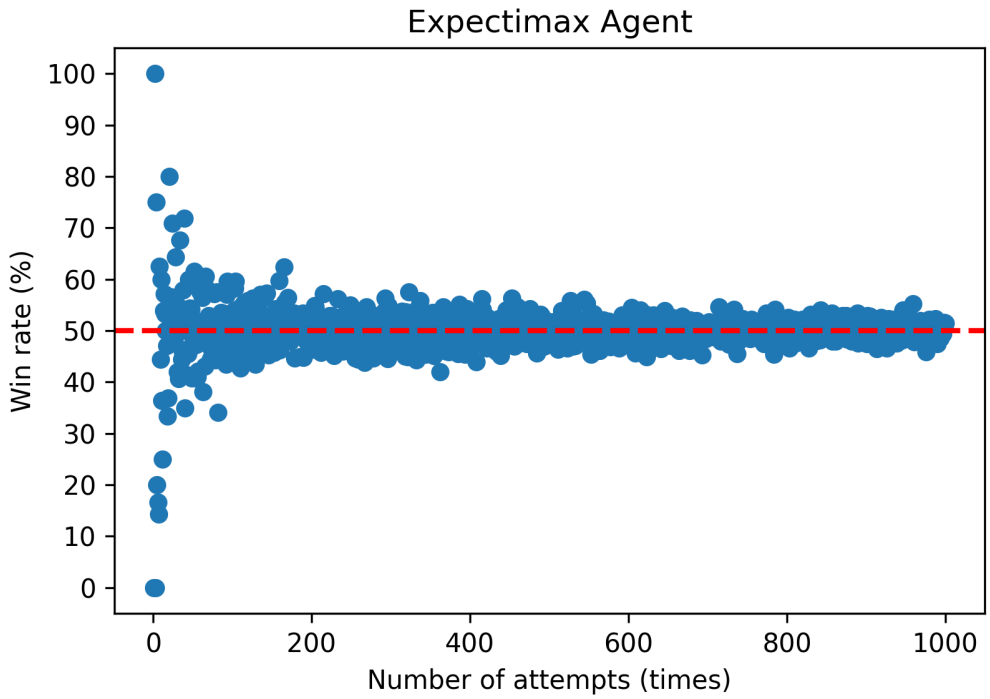
  쉘 스크립트를 통해 Expectimax Agent를 1번 ~ 1000번의 게임을 자동으로 진행하게 하였습니다.

```
#!/bin/bash

for var in {1..1000}
do
```

```
  python3 pacman.py -p ExpectimaxAgent -m stuckmap -a depth=5 -n $var -q
done
```

- **결과**
  - 각 게임의 Win rate를 수집하고 matplotlib을 사용하여 아래와 같이 시각화하였습니다.



시도 횟수(Number of attempts)가 커짐에 따라 승률(Win rate)이 50%에 가까이 분포하고 있다는 것을 볼 수 있습니다.