AI Assignment02 보고서

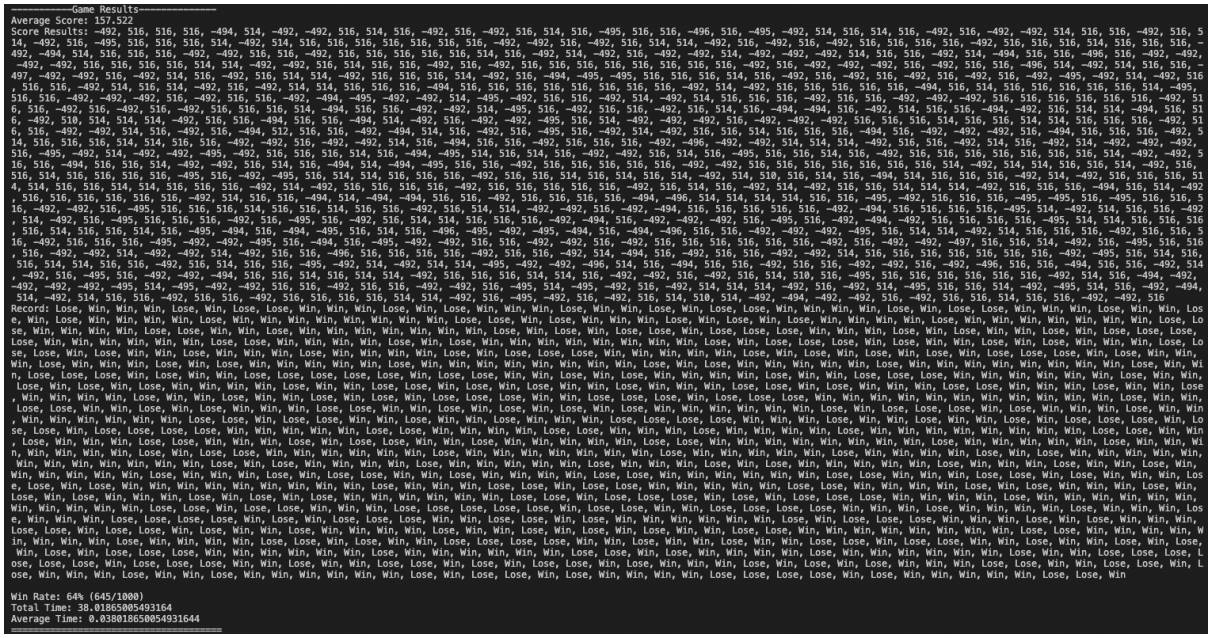
```
AI Assignment02 보고서
학번 : 20171612
학과 : 컴퓨터공학과
이름 : 김성일
```

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"]
    ##############################################################################
```

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.

Maximize하는 경우

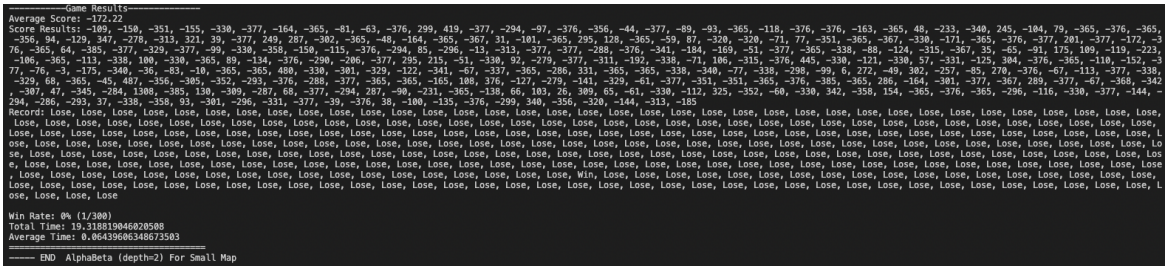
```
if maximizing:
  score = self.alpha_beta(new_state, depth, 1, False, alpha, beta)["score"]
  result["action"], result["score"] = [(result["action"], result["score"]), (action, score)][score > result["score"]]
  # Pruning
  if result["score"] >= beta:
    break
  alpha = max(alpha, result["score"])
```

Minimize하는 경우

```
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
```

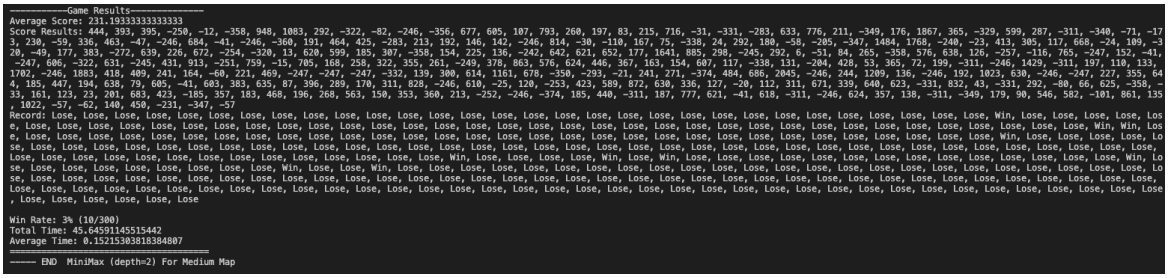
결과

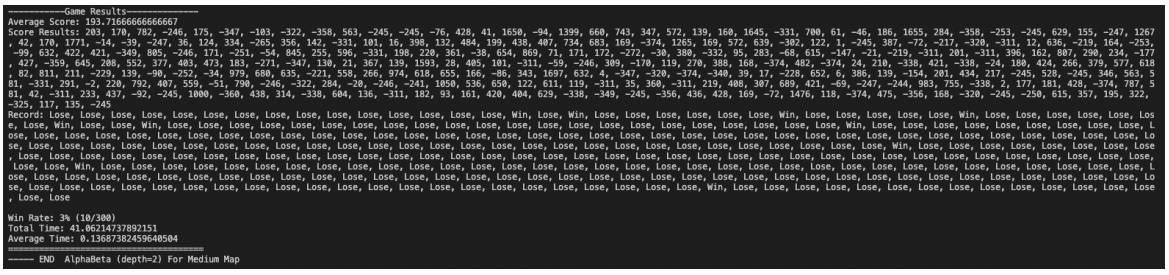
- **Small Map**



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

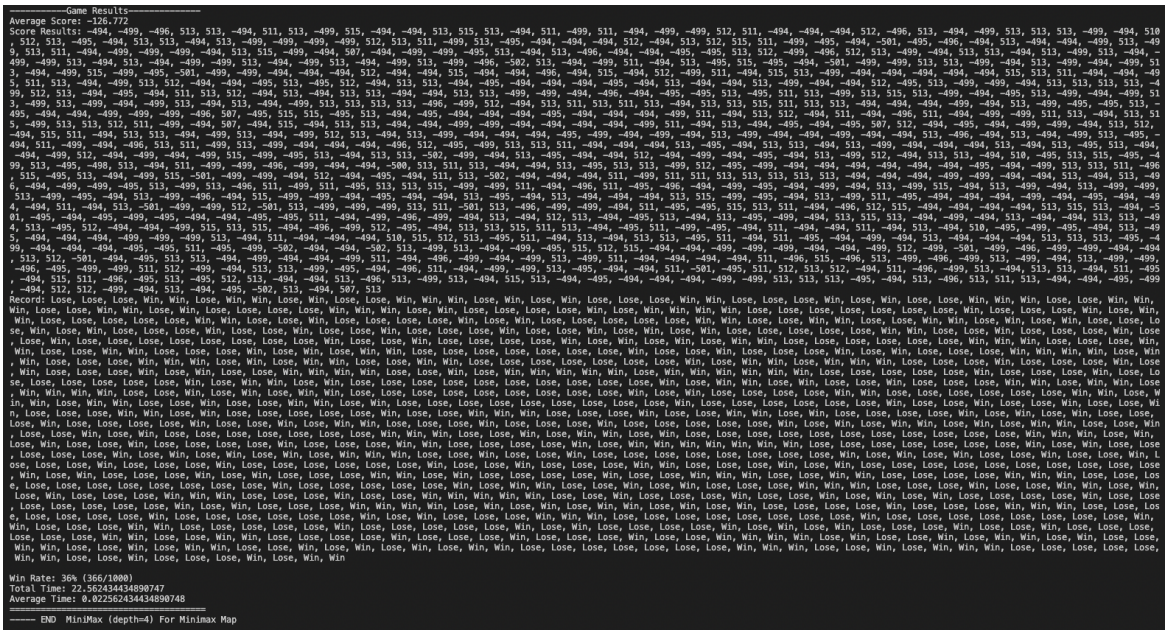
- **Medium Map**

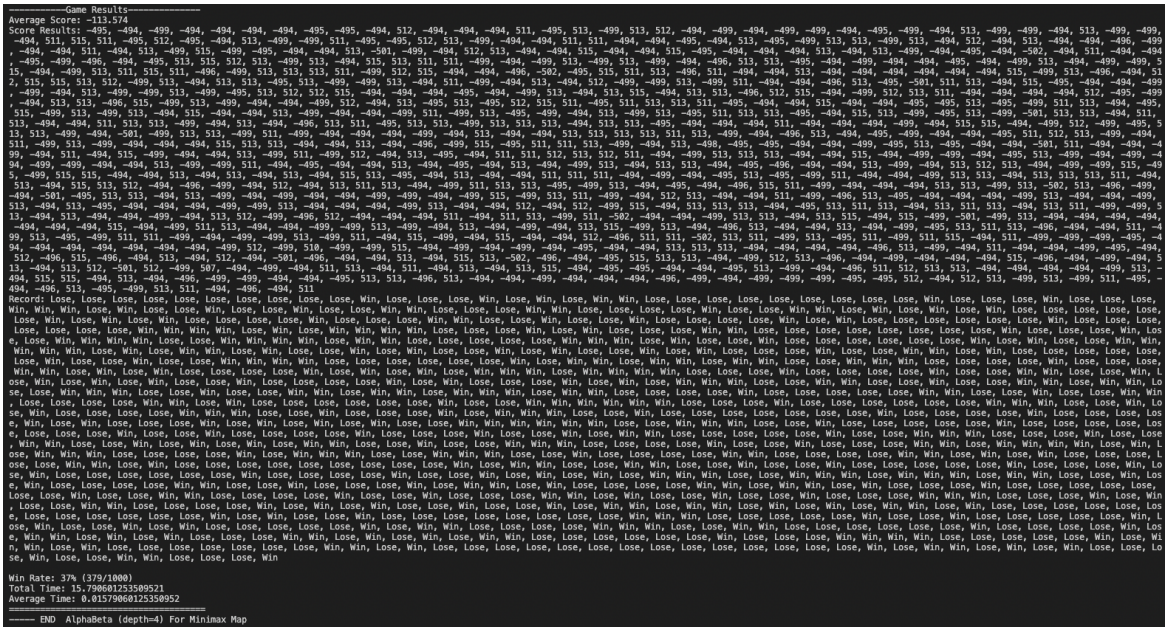




- **MiniMax** Average Execute Time : 0.1521530 sec
- **AlphaBeta** Average Execute Time : 0.136873sec

- **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

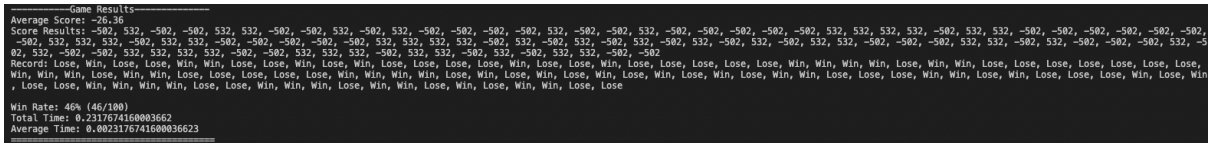
코드

```
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 expactimax(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.expactimax(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.expactimax(new_state, depth + 1, 0, True)["score"]
        else:
          score = self.expactimax(new_state, depth, agent + 1, False)["score"]
        result["score"] += prob(score)
    return result

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

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%정도가 나오며, 50% 정도**임을 알 수 있습니다.

1~ 1000번의 게임을 실행했을 때 승률의 분포

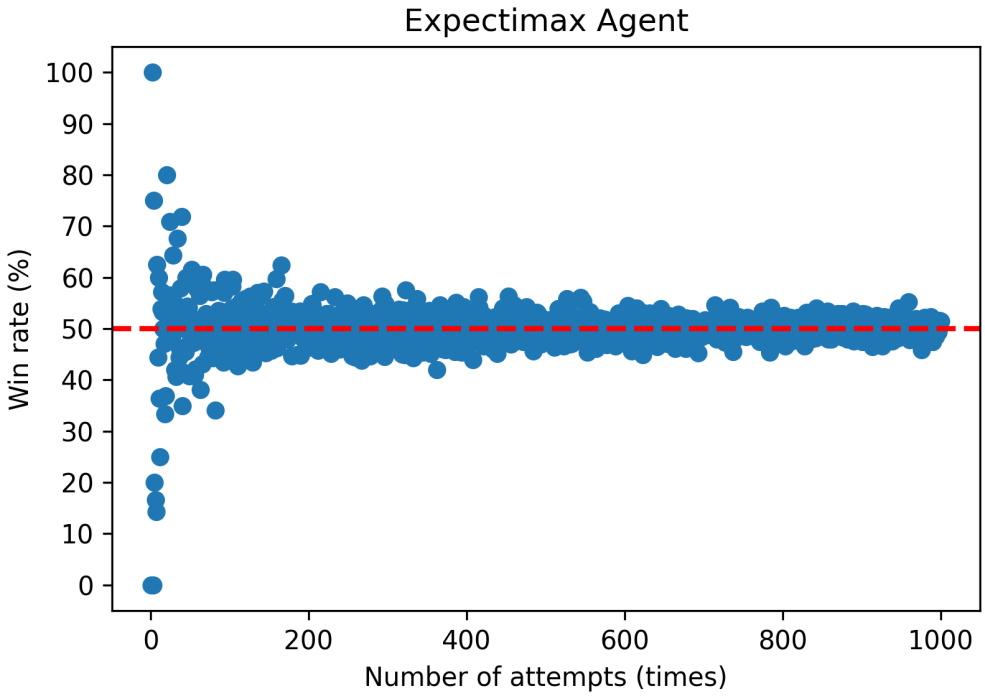
이것을 더 확실하게 살펴보기 위해 1번부터 1000번까지 각각의 게임을 진행해보았습니다.

- **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.