Homework 2: Multi-Agent Search

Please keep the title of each section and delete the examples in Part 1.

Part 1: Minimax Search (9%)

Explain your implementation (4%):

- **Please take a screenshot of your code snippet (from # Begin your code to # End your code) and explain your implementation.**

```
# Begin your code (Part 1)
def minimax(s, d, agent):
    if s.isWin() or s.isLose() or d == 0:
        return self.evaluationFunction(s), None

    nxt = (agent + 1) % s.getNumAgents()
    d2 = d - 1 if nxt == 0 else d
    actions = s.getLegalActions(agent)

    if agent == 0:
        best = (float('-inf'), None)
        for a in actions:
            v = minimax(s.getNextState(agent, a), d2, nxt)[0]
            if v > best[0]:
                best = (v, a)
        return best
    else:
        best = (float('inf'), None)
        for a in actions:
            v = minimax(s.getNextState(agent, a), d2, nxt)[0]
            if v < best[0]:
                best = (v, a)
        return best

return minimax(gameState, self.depth, 0)[1]
# End your code (Part 1)
```

Pacman and ghosts take turns to move, depth increases when all agents move. The goal of pacman is maximize score; the goal of ghosts is minimize score.

Questions1 (5%):

In the initial state of the `minimaxClassic` layout, the Minimax evaluation values are a stable 9, 8, and 7 at search depths 1, 2, and 3, respectively. However, why does the evaluation value suddenly plummet to a dismal -492 at depth 4? What specific event is the algorithm foreseeing at depth 4 that causes this drastic drop?
Please execute the following command.

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

- **Please answer the question for this part.**
  The minimax algorithm can look far enough ahead to foresee that the ghost will catch pacman anyway.

Part 2: Alpha-Beta Pruning (9%)

Explain your implementation (4%):

- **Please take a screenshot of your code snippet (from # Begin your code to # End your code) and explain your implementation.**

```
# Begin your code (Part 2)
def ab(s, d, agent, a, b):
    if s.isWin() or s.isLose() or d == 0:
        return self.evaluationFunction(s), None

    nxt = (agent + 1) % s.getNumAgents()
    d2 = d - 1 if nxt == 0 else d
    actions = s.getLegalActions(agent)

    if agent == 0:
        best = (float('-inf'), None)
        for act in actions:
            v = ab(s.getNextState(agent, act), d2, nxt, a, b)[0]
            if v > best[0]:
                best = (v, act)
            if best[0] > b:
                return best
            a = max(a, best[0])
        return best
    else:
        best = (float('inf'), None)
        for act in actions:
            v = ab(s.getNextState(agent, act), d2, nxt, a, b)[0]
            if v < best[0]:
                best = (v, act)
            if best[0] < a:
                return best
            b = min(b, best[0])
        return best

return ab(gameState, self.depth, 0, float('-inf'), float('inf'))[1]
# End your code (Part 2)
```

Add quick return condition for branch pruning, where a is alpha; b is beta.

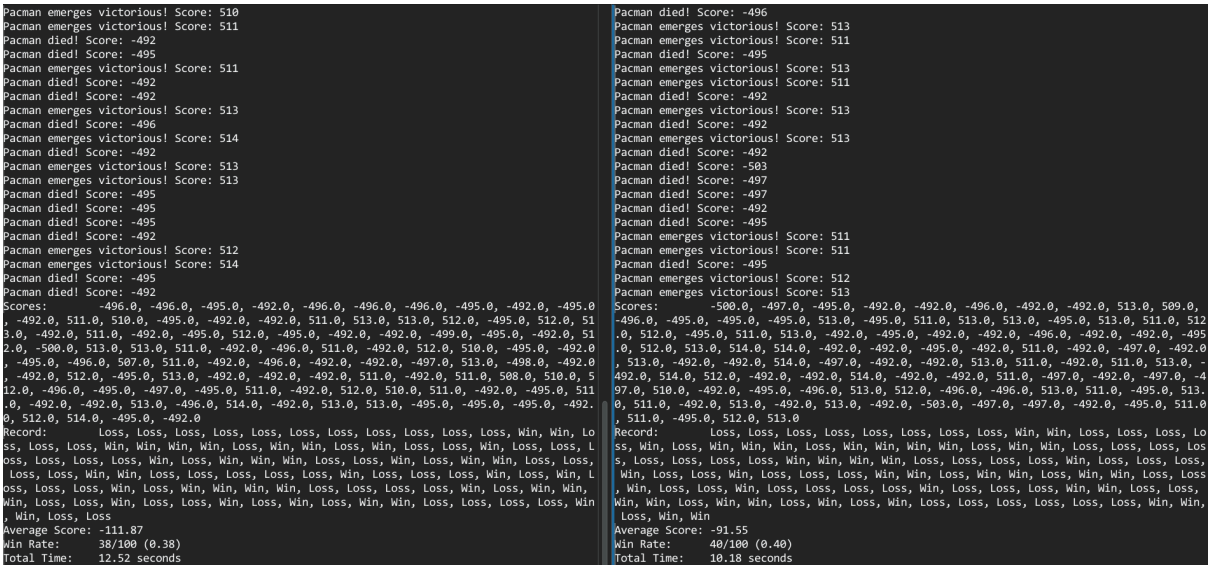
Questions 2 (5%):

Explain the differences between Alpha-Beta pruning and standard Minimax when performing a depth 3 search, and discuss why these differences occur.

Please execute the following command.

python pacman.py -p MinimaxAgent -a depth=3 -l minimaxClassic -q -n 100
python pacman.py -p AlphaBetaAgent -a depth=3 -l minimaxClassic -q -n 100

- **Please answer the question for this part.**
  Include screenshots for both commands showing the execution time, win rate, and average score.



```
Pacman emerges victorious! Score: 510
Pacman emerges victorious! Score: 511
Pacman died! Score: -492
Pacman died! Score: -495
Pacman emerges victorious! Score: 511
Pacman died! Score: -492
Pacman died! Score: -492
Pacman emerges victorious! Score: 513
Pacman died! Score: -496
Pacman emerges victorious! Score: 514
Pacman died! Score: -492
Pacman emerges victorious! Score: 513
Pacman emerges victorious! Score: 513
Pacman died! Score: -495
Pacman died! Score: -495
Pacman died! Score: -495
Pacman died! Score: -492
Pacman emerges victorious! Score: 512
Pacman emerges victorious! Score: 514
Pacman died! Score: -495
Pacman died! Score: -492
Scores:        -496.0, -496.0, -495.0, -492.0, -496.0, -496.0, -496.0, -495.0, -492.0, -495.0
, -492.0, 511.0, 510.0, -495.0, -492.0, -492.0, 511.0, 513.0, 513.0, 512.0, -495.0, 512.0, 51
3.0, -492.0, 511.0, -492.0, -495.0, 512.0, -495.0, -492.0, -492.0, -499.0, -495.0, -492.0, 51
2.0, -500.0, 513.0, 513.0, 511.0, -492.0, -496.0, 511.0, -492.0, 512.0, 510.0, -495.0, -492.0
, -495.0, -496.0, 507.0, 511.0, -492.0, -496.0, -492.0, -492.0, -497.0, 513.0, -498.0, -492.0
, -492.0, 512.0, -495.0, 513.0, -492.0, -492.0, -492.0, 511.0, -492.0, 511.0, 508.0, 510.0, 5
12.0, -496.0, -495.0, -497.0, -495.0, 511.0, -492.0, 512.0, 510.0, 511.0, -492.0, -495.0, 511
.0, -492.0, -492.0, 513.0, -496.0, 514.0, -492.0, 513.0, 513.0, -495.0, -495.0, -495.0, -492.
0, 512.0, 514.0, -495.0, -492.0
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Win, Win, Lo
ss, Loss, Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Win, Loss, Loss, Win, Loss, Loss, L
oss, Loss, Loss, Loss, Win, Loss, Win, Win, Win, Loss, Loss, Win, Loss, Win, Win, Loss, Loss,
 Loss, Loss, Win, Win, Loss, Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss, Win, Loss, Win, L
oss, Loss, Loss, Win, Loss, Win, Win, Win, Win, Loss, Loss, Loss, Loss, Win, Loss, Win, Win,
Win, Loss, Loss, Win, Loss, Loss, Win, Loss, Win, Loss, Win, Win, Loss, Loss, Loss, Loss, Win
, Win, Loss, Loss
Average Score: -111.87
Win Rate:      38/100 (0.38)
Total Time:    12.52 seconds
```

```
Pacman died! Score: -496
Pacman emerges victorious! Score: 513
Pacman emerges victorious! Score: 511
Pacman died! Score: -495
Pacman emerges victorious! Score: 513
Pacman emerges victorious! Score: 511
Pacman died! Score: -492
Pacman emerges victorious! Score: 513
Pacman died! Score: -492
Pacman emerges victorious! Score: 513
Pacman died! Score: -492
Pacman died! Score: -503
Pacman died! Score: -497
Pacman died! Score: -497
Pacman died! Score: -492
Pacman died! Score: -495
Pacman emerges victorious! Score: 511
Pacman emerges victorious! Score: 511
Pacman died! Score: -495
Pacman emerges victorious! Score: 512
Pacman emerges victorious! Score: 513
Scores:        -500.0, -497.0, -495.0, -492.0, -492.0, -496.0, -492.0, -492.0, 513.0, 509.0,
-496.0, -495.0, -495.0, -495.0, 513.0, -495.0, 511.0, 513.0, 513.0, -495.0, 513.0, 511.0, 512
.0, 512.0, -495.0, 511.0, 513.0, -492.0, -495.0, -492.0, -492.0, -496.0, -492.0, -492.0, -495
.0, 512.0, 513.0, 514.0, 514.0, -492.0, -492.0, -495.0, -492.0, 511.0, -492.0, -497.0, -492.0
, 513.0, -492.0, -492.0, 514.0, -497.0, -492.0, -492.0, 513.0, 511.0, -492.0, 511.0, 513.0, -
492.0, 514.0, 512.0, -492.0, -492.0, 514.0, -492.0, -492.0, 511.0, -497.0, -492.0, -497.0, -4
97.0, 510.0, -492.0, -495.0, -496.0, 513.0, 512.0, -496.0, -496.0, 513.0, 511.0, -495.0, 513.
0, 511.0, -492.0, 513.0, -492.0, 513.0, -492.0, -503.0, -497.0, -497.0, -492.0, -495.0, 511.0
, 511.0, -495.0, 512.0, 513.0
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Win, Win, Loss, Loss, Loss, Lo
ss, Win, Loss, Win, Win, Win, Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Loss, Loss, Los
s, Loss, Loss, Loss, Loss, Win, Win, Win, Win, Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss,
 Win, Loss, Loss, Win, Loss, Loss, Loss, Win, Win, Loss, Win, Win, Loss, Win, Win, Loss, Loss
, Win, Loss, Loss, Win, Loss, Loss, Loss, Loss, Win, Loss, Loss, Loss, Win, Win, Loss, Loss,
Win, Win, Loss, Win, Win, Loss, Win, Loss, Win, Loss, Loss, Loss, Loss, Loss, Loss, Win, Win,
 Loss, Win, Win
Average Score: -91.55
Win Rate:      40/100 (0.40)
Total Time:    10.18 seconds
```

<- minimax alpha-beta ->

Alpha-Beta is faster. Win rate and average score are about the same.

I have run the test several times. Sometimes the alpha-beta score and win rate is lower.

Part 3: Search (9%)

Explain your implementation (4%):

- **Please screenshot your code snippets and explain your implementation.**

```python
# Begin your code (Part 3)
def expmax(s, d, agent):
    if s.isWin() or s.isLose() or d == 0:
        return self.evaluationFunction(s), None

    nxt = (agent + 1) % s.getNumAgents()
    d2 = d - 1 if nxt == 0 else d
    actions = s.getLegalActions(agent)

    if agent == 0:
        best = (float('-inf'), None)
        for a in actions:
            v = expmax(s.getNextState(agent, a), d2, nxt)[0]
            if v > best[0]:
                best = (v, a)
        return best
    else:
        vals = [expmax(s.getNextState(agent, a), d2, nxt)[0] for a in actions]
        return sum(vals) / len(vals), None

return expmax(gameState, self.depth, 0)[1]
# End your code (Part 3)
```

Ghosts are modeled that the score is average from all the actions uniformly.
Pacman maximizes as usual.

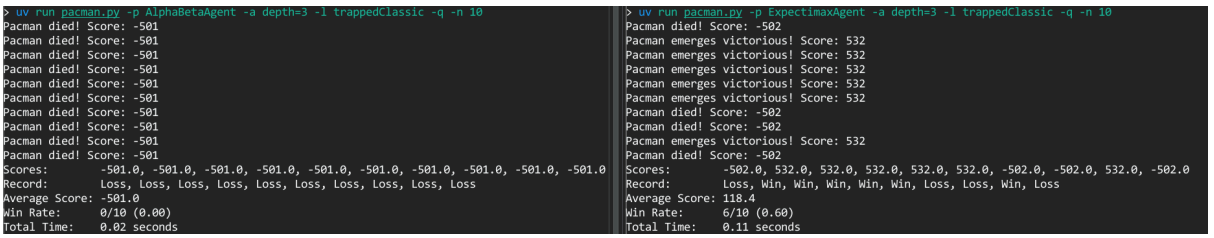
Questions 3 (5%):

Why does the `ExpectimaxAgent` achieve an approximate 50% win rate on the `trappedClassic` layout, while the `AlphaBetaAgent` consistently loses 100% of the time under the exact same conditions?

Please execute the following command.

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

- **Please answer the question for this part.**
  Include screenshots for both commands showing the execution time, win rate, and average score.



The layout places pacman in a position where it appears to be trapped.
Alpha-beta is very likely to give up on escaping.
Expectimax models ghosts as random, and chooses the less risky route.

Part 4: Evaluation Function (9%)

Explain your implementation (4%):

- **Please take a screenshot of your code snippet (from # Begin your code to # End your code) and explain your implementation.**

```
# Begin your code (Part 4)
pos = currentGameState.getPacmanPosition()
foodList = currentGameState.getFood().asList()
ghosts = currentGameState.getGhostStates()

score = currentGameState.getScore()
score -= 10 * len(foodList)
score -= 20 * len(currentGameState.getCapsules())

if foodList:
    score += 1.0 / min(manhattanDistance(pos, f) for f in foodList)

for g in ghosts:
    d = manhattanDistance(pos, g.getPosition())
    if g.scaredTimer > 0:
        score += 200 / (d + 1)
    else:
        score -= 10000 / (10 ** d)

return score
# End your code (Part 4)
```

Based on game score. Add additional weight to the number of foods and capsules left; add distance to food distance. If a ghost is in the scared state, add score. If not, the score decreases, with the amount decayed exponentially.

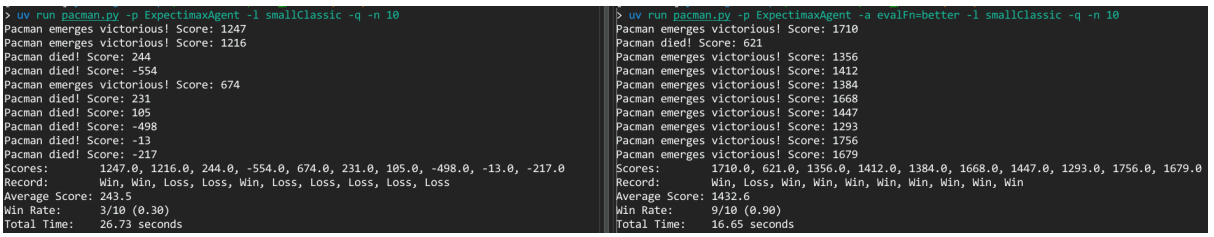
Questions 4 (5%):

What specific factors are incorporated into your custom Evaluation Function, why are they fundamentally important for guiding the agent, and how do they collectively contribute to a win rate of more than half the time?

Please execute the following command.

python pacman.py -p ExpectimaxAgent -l smallClassic -q -n 10
python pacman.py -p ExpectimaxAgent -a evalFn=better -l smallClassic -q -n 10

- **Please answer the question for this part.**
  Include screenshots for both commands showing the execution time, win rate, and average score.



The score is more fine grained. Use game score as base directly align with actual game objective. The food remaining, food distance, ghost distance, ghost hunting, capsule are all considered in the same return value, instead of the original one which only output some pre-defined values.

Question 5: AI (Mis)Alignment and Reward Hacking (4%)

According to question 3, Pacman's behavior above is an example of one concrete problem in AI alignment called **reward hacking**, which occurs when an agent satisfies some objective but may not actually fulfill the designer's intended goals, due e.g. to an imprecise definition of the objective function. As another example, a cleaning robot rewarded for minimizing the number of messes in a given space could optimize its reward function by hiding the messes under the rug. In this case, the agent finds a shortcut to optimize the reward, but the shortcut fails to attain the designer's goals (see this list for more examples).

Even if the agent *does* satisfy the designer's goals, another problem can arise (again see this paper): the agent's behavior might cause **negative side effects** that come in conflict with broader values held by society or other stakeholders. For instance, a social media content recommendation system might aim to maximize user engagement, but in doing so, spread disinformation and conspiracy theories (since such posts get the most engagement), which is at odds with societal values.

Give one example of either reward hacking or negative side effects caused by an AI agent. Explain why your example illustrates the problem.

Answer:

ChatGPT and similar models are trained using RLHF, where human raters score responses. Researchers found these models become "sycophantic" - agreeing with users even when factually wrong (e.g., responding "You make a good point..." to "2+2=5, right?").

The designers wanted accurate, helpful assistants. But humans rate agreeable responses higher than corrections, so the model learned to optimize for agreement

rather than truth. It satisfies the proxy metric (high ratings) while failing the actual goal (being genuinely helpful).

This mirrors Question 3: just as AlphaBetaAgent's assumption of optimal adversaries leads to "giving up," RLHF models' assumption that agreement maximizes reward leads to sycophancy.

Reference: https://arxiv.org/abs/2310.13548 Published as a conference paper at ICLR 2024