HW3 report

Part I. Implementation (20%):

- Part 1

```python
    # Begin your code (Part 1)
    #raise NotImplementedError("To be implemented")
    def value(agent, state, depth): # Calculate the value of each node
        if state.isWin() or state.isLose() or depth == 0: # If game is over or reach the depth, return current value
            return self.evaluationFunction(state)

        next = (agent + 1) % state.getNumAgents() # Calculate next agent index
        if next == 0: # If next is zero, which means a single level search is done
            depth -= 1

        if agent == 0: # If agent is pacman, calculate its max value from its successors
            v = float("-inf")
            for a in state.getLegalActions(agent):
                v = max(v, value(next, state.getNextState(agent, a), depth))
            return v
        else: # If agent is ghost, calculate its min value from its successors
            v = float("inf")
            for a in state.getLegalActions(agent):
                v = min(v, value(next, state.getNextState(agent, a), depth))
            return v

    maximum = float("-inf")
    action = None
    for a in gameState.getLegalActions(0): # First agent is pacman, calculate its max value to get best action
        v = value(1, gameState.getNextState(0, a), self.depth)
        if v > maximum:
            maximum = v
            action = a

    return action

    # End your code (Part 1)
```

(parameter) depth: Any

- Part 2

```
183         def value(agent, state, depth, alpha, beta): # Calculate the value of each node
184             if state.isWin() or state.isLose() or depth == 0: # If game is over or reach the depth, return current value
185                 return self.evaluationFunction(state)
186 
187             next = (agent + 1) % state.getNumAgents() # Calculate next agent index
188             if next == 0: # If next is zero, which means a single level search is done
189                 depth -= 1
190 
191             if agent == 0: # If agent is pacman, calculate its max value from its successors
192                 v = float("-inf")
193                 for a in state.getLegalActions(agent):
194                     v = max(v, value(next, state.getNextState(agent, a), depth, alpha, beta))
195                     if v > beta: # If value is greater than beta, do pruning
196                         return v
197                     alpha = max(v, alpha) # Update alpha
198                 return v
199             else: # If agent is ghost, calculate its min value from its successors
200                 v = float("inf")
201                 for a in state.getLegalActions(agent):
202                     v = min(v, value(next, state.getNextState(agent, a), depth, alpha, beta))
203                     if v < alpha: # If value is smaller than alpha, do pruning
204                         return v
205                     beta = min(v, beta) # Update beta
206                 return v
207 
208         maximum = float("-inf")
209         action = None
210         alpha = float("-inf")
211         beta = float("inf")
212         for a in gameState.getLegalActions(0): # First agent is pacman, calculate its max value to get best action
213             v = value(1, gameState.getNextState(0, a), self.depth, alpha, beta)
214             if v > maximum:
215                 maximum = v
216                 action = a
217             # Value must be smaller than beta, so do not need pruning
218             alpha = max(maximum, alpha) # Update alpha
219 
220         return action
221         # End your code (Part 2)
```

- Part 3

```
240         def value(agent, state, depth): # Calculate the value of each node
241             if state.isWin() or state.isLose() or depth == 0: # If game is over or reach the depth, return current value
242                 return self.evaluationFunction(state)
243 
244             next = (agent + 1) % state.getNumAgents() # Calculate next agent index
245             if next == 0: # If next is zero, which means a single level search is done
246                 depth -= 1
247 
248             if agent == 0: # If agent is pacman, calculate its max value from its successors
249                 v = float("-inf")
250                 for a in state.getLegalActions(agent):
251                     v = max(v, value(next, state.getNextState(agent, a), depth))
252                 return v
253             else: # If agent is ghost, calculate its mean value from its successors
254                 v = 0
255                 for a in state.getLegalActions(agent):
256                     v += value(next, state.getNextState(agent, a), depth)
257                 return float(v / len(state.getLegalActions(agent)))
258 
259         maximum = float("-inf")
260         action = None
261         for a in gameState.getLegalActions(0): # First agent is pacman, calculate its max value to get best action
262             v = value(1, gameState.getNextState(0, a), self.depth)
263             if v > maximum:
264                 maximum = v
265                 action = a
266 
267         return action
268         # End your code (Part 3)
```

- Part 4

```
    # Begin your code (Part 4)
    #raise NotImplementedError("To be implemented")
    Pos = currentGameState.getPacmanPosition() # Get pacman position
    GhostStates = currentGameState.getGhostStates() # Get ghosts states
    Food = currentGameState.getFood() # Get food position
    Capsule = currentGameState.getCapsules() # Get capsules position
    numFood = currentGameState.getNumFood() # Get number of food
    numCapsules = len(Capsule) # Get number of capsules

    score = currentGameState.getScore() # Get current score
    scare = 0 # A flag to show whether a ghost is scared

    minGhostDistance = min([manhattanDistance(Pos, state.getPosition()) for state in GhostStates]) # Get closest ghost distance
    minCapsuleDistance = 0
    if numCapsules > 0: # If there exist capsule, get the closest capsule distance
        minCapsuleDistance = min([manhattanDistance(Pos, capsule) for capsule in Capsule])
    minScareDistance = float("inf")
    for state in GhostStates:
        if state.scaredTimer > 0: # If there exist scared ghost, set scare true, and get closest scared ghost distance
            minScareDistance = min(minScareDistance, manhattanDistance(Pos, state.getPosition()))
            scare = 1

    nearestFoodDistance = min([manhattanDistance(Pos, food) for food in Food]) # Get closest food distance

    if scare: # If some ghosts are scared, return the value below
        return 10 * score - 50 * minScareDistance
    elif minGhostDistance > 3: # If no ghost near pacman by 3 steps, return the value below
        return 10 * score + (-10 * nearestFoodDistance) + (-20 * minCapsuleDistance) + (-20 * numFood) + (-50 * numCapsules)
    else: # Else, return the value below
        return 10 * score + (10 * minGhostDistance) + (-1 * nearestFoodDistance) + (-20 * minCapsuleDistance) + (-10 * numFood) + (-25 * numCapsules)
    # End your code (Part 4)
```

Part II. Results & Analysis (10%):

```
Average Score: 1232.5
Scores:        1254.0, 1265.0, 1121.0, 1303.0, 1202.0, 1187.0, 1279.0, 1267.0, 1211.0, 1236.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\part4\grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
***     1232.5 average score (4 of 4 points)
***         Grading scheme:
***          < 600:  0 points
***         >= 600:  2 points
***         >= 1200:  4 points
***     10 games not timed out (2 of 2 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 5:  1 points
***         >= 10:  2 points
***     10 wins (4 of 4 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 4:  2 points
***         >= 7:  3 points
***         >= 10:  4 points

### Question part4: 10/10 ###


Finished at 17:34:02

Provisional grades
==================
Question part1: 15/15
Question part2: 20/20
Question part3: 20/20
Question part4: 10/10
------------------
Total: 65/65
```

- In part 4, I used 8 parameters to construct my better evaluation function, and I fine-tuned their coefficients and composition for about 4 hours to get the average score above 1200. My main strategy is when some ghosts are scared, I give minScareDistance coefficient -50 to let pacman get closer to scared ghosts. If no ghost is scared and minGhostDistance > 3, I let pacman randomly eat food and capsules. If there are some ghosts that are too close, I give minGhostDistance coefficient 10 to let pacman get away from ghosts. I randomly selected some random seed to test my better evaluation function, and I noticed that it performed not bad. It sometimes would die once or average score is only 11XX, but it usually could get full points.