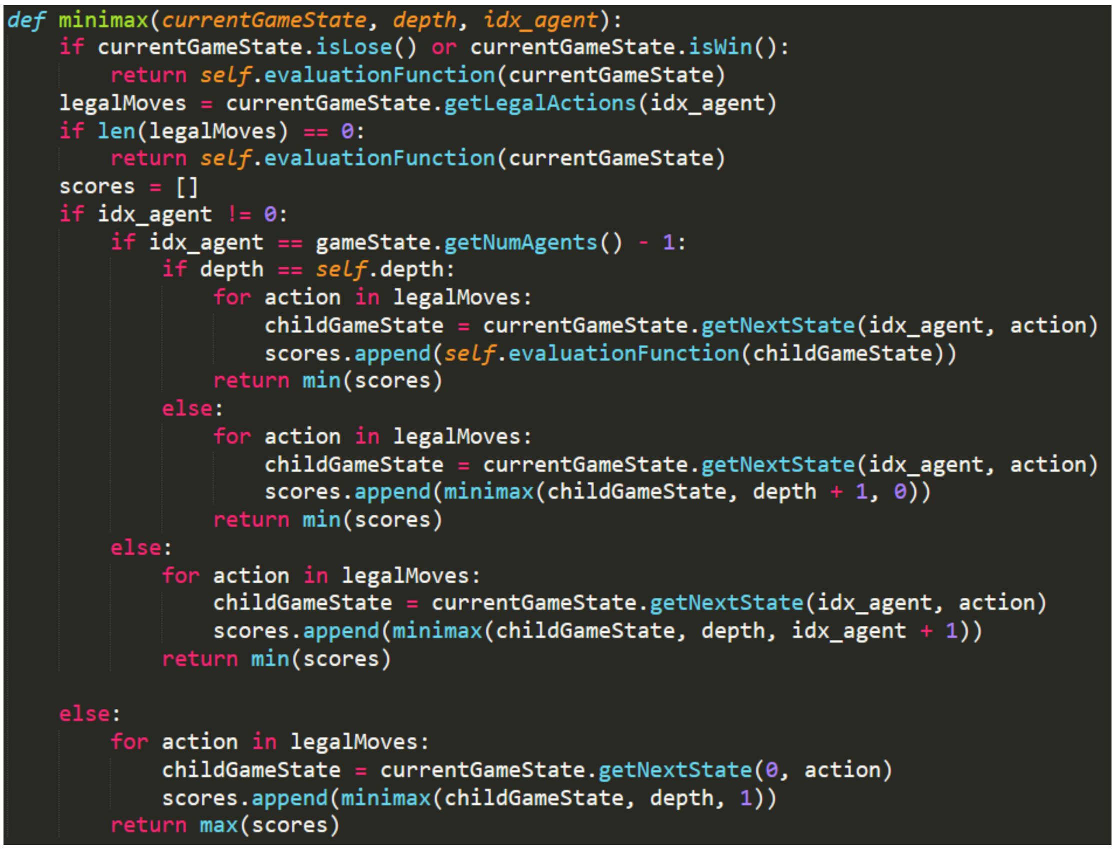
# HW3 Multi-Agent Search

0616098 黃秉茂

## Part 1



```python
def minimax(currentGameState, depth, idx_agent):
    if currentGameState.isLose() or currentGameState.isWin():
        return self.evaluationFunction(currentGameState)
    legalMoves = currentGameState.getLegalActions(idx_agent)
    if len(legalMoves) == 0:
        return self.evaluationFunction(currentGameState)
    scores = []
    if idx_agent != 0:
        if idx_agent == gameState.getNumAgents() - 1:
            if depth == self.depth:
                for action in legalMoves:
                    childGameState = currentGameState.getNextState(idx_agent, action)
                    scores.append(self.evaluationFunction(childGameState))
                return min(scores)
            else:
                for action in legalMoves:
                    childGameState = currentGameState.getNextState(idx_agent, action)
                    scores.append(minimax(childGameState, depth + 1, 0))
                return min(scores)
        else:
            for action in legalMoves:
                childGameState = currentGameState.getNextState(idx_agent, action)
                scores.append(minimax(childGameState, depth, idx_agent + 1))
            return min(scores)

    else:
        for action in legalMoves:
            childGameState = currentGameState.getNextState(0, action)
            scores.append(minimax(childGameState, depth, 1))
        return max(scores)
```

minimax is the recursive function which will return the biggest score and help for choosing the action which pacman should take, and it assumes the worst condition that all the agent will act their optimal behavior.

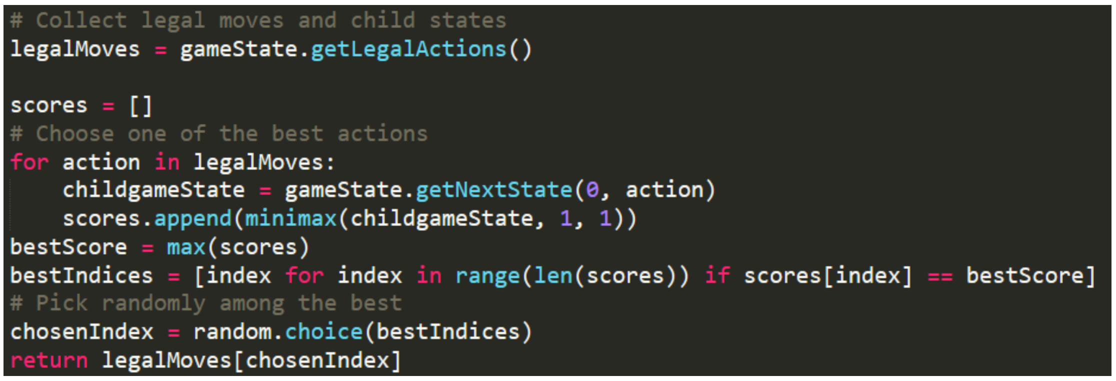
Value of a state: The best achievable outcome (utility) from that state.

If isLose() or isWin() is True, it represents that there is no any child state and it is a terminal state. Thus, it should return the score of the state.

If the length of legalMoves is zero, it means that there isn't any legal action. So, there is no any child state and it is a terminal state. Therefore, it should return the score of the state.

The list scores will record the list of scores in this iteration, and we want to choose the max or min of that accoding to the condition.

idx_agent is 0 when the agent is the pacman, or idx_agent is more than 0 when the agent is a ghost.
The child states represent the state which the current state will become when taking its legal actions. It is generated by getNextState with the current state.
The ghost will enter the min state and want to find the min score in this iteration because it want to worsen the pacman. So it appends all the scores determined by all the child states and the next agent into the list scores, and pick the smallest one to return.
If the idx_agent equals (getNumAgents() - 1), it means it is the final ghost to discuss in this depth. Thus, it should go to the next depth and consider the pacman again. However, it should be noticed that if it is also at the bottom depth. If so, it is the parents of the leaf and it should consider the immediate score of its child states intead of going to the next depth or considering the pacman.
The pacman will enter the max state and want to find the max score in this iteration because it want get the largest score. So it appends all the scores determined by all the child states and the next agent (1st ghost) into the list scores, and pick the biggest one to return.

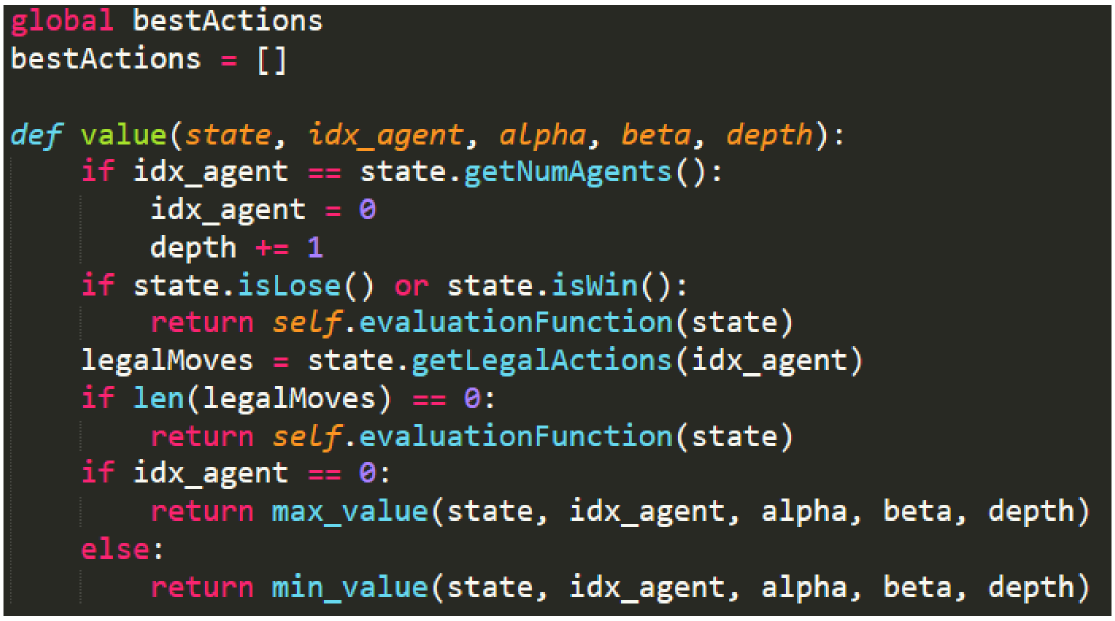


```
# Collect legal moves and child states
legalMoves = gameState.getLegalActions()

scores = []
# Choose one of the best actions
for action in legalMoves:
    childgameState = gameState.getNextState(0, action)
    scores.append(minimax(childgameState, 1, 1))
bestScore = max(scores)
bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
# Pick randomly among the best
chosenIndex = random.choice(bestIndices)
return legalMoves[chosenIndex]
```

First, we get the all legal actions of the state.
The list scores will record the list of scores determined by the state taking different legal actions.
We collect the different scores when the state take different lagal actions.
Next, calculate the max score and find the position of the action which will lead to the biggest score.
Finally, we find the best action which will lead the pacman to the largest score and choose one of them to return.

Part 2

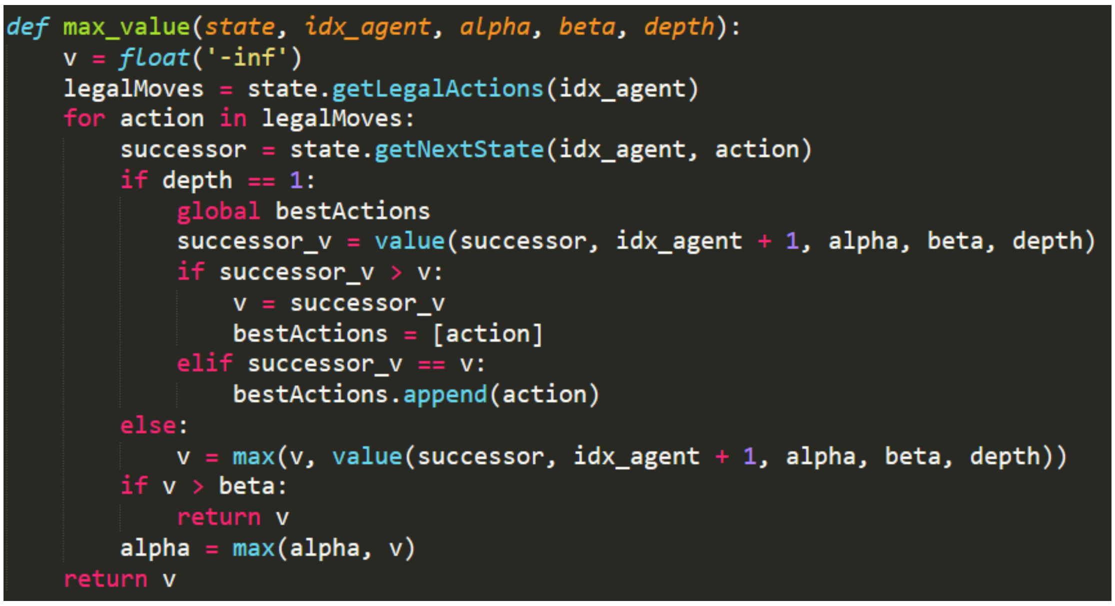


```
global bestActions
bestActions = []

def value(state, idx_agent, alpha, beta, depth):
    if idx_agent == state.getNumAgents():
        idx_agent = 0
        depth += 1
    if state.isLose() or state.isWin():
        return self.evaluationFunction(state)
    legalMoves = state.getLegalActions(idx_agent)
    if len(legalMoves) == 0:
        return self.evaluationFunction(state)
    if idx_agent == 0:
        return max_value(state, idx_agent, alpha, beta, depth)
    else:
        return min_value(state, idx_agent, alpha, beta, depth)
```

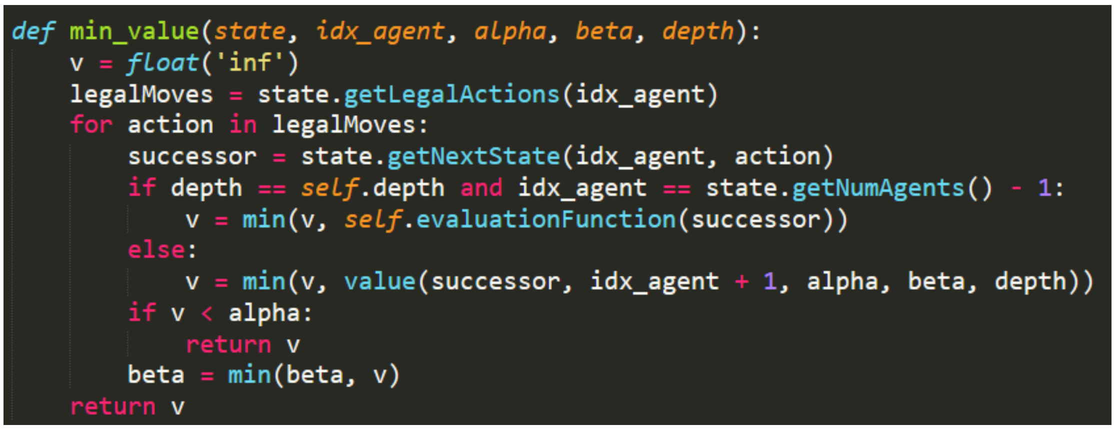
Alpha-Beta Pruning is almost same as minimax. It is also the recursive function which will return the biggest score and help for choosing the action which pacman should take, and it assumes the worst condition that all the agent will act their optimal behavior. The only difference between them is that Alpha-Beta Pruning is more efficient, since it would prune some branchs which we will impossible to reach.
Value of a state: The best achievable outcome (utility) from that state.
bestActions is a list which record the actions which can bring us to the highest score.
value function return the score of the state.
If the idx_agent equals (getNumAgents() - 1), it means it is the final ghost to discuss in this depth. Thus, it should go to the next depth and consider the pacman again.
If isLose() or isWin() is True, it represents that there is no any child state and it is a terminal state. Thus, it should return the score of the state.
If the length of legalMoves is zero, it means that there isn’t any legal action. So, there is no any child state and it is a terminal state. Therefore, it should return the score of the state.
The list scores will record the list of scores in this iteration, and we want to choose the max or min of that accoding to the condition.
idx_agent is 0 when the agent is the pacman, or idx_agent is more than 0 when the agent is a ghost.
The ghost will enter the min state and want to find the min score in this iteration because it want to worsen the pacman.
The pacman will enter the max state and want to find the max score in this iteration because it want get the largest score.

All the state would be pruning in their callee function if it is necessary.
The function decide what the value of the state should be returned.



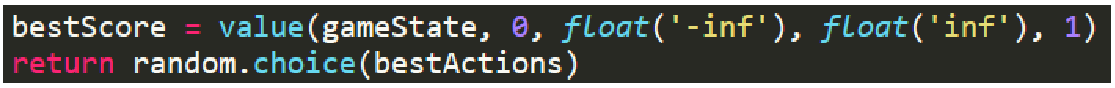
```python
def max_value(state, idx_agent, alpha, beta, depth):
    v = float('-inf')
    legalMoves = state.getLegalActions(idx_agent)
    for action in legalMoves:
        successor = state.getNextState(idx_agent, action)
        if depth == 1:
            global bestActions
            successor_v = value(successor, idx_agent + 1, alpha, beta, depth)
            if successor_v > v:
                v = successor_v
                bestActions = [action]
            elif successor_v == v:
                bestActions.append(action)
        else:
            v = max(v, value(successor, idx_agent + 1, alpha, beta, depth))
        if v > beta:
            return v
        alpha = max(alpha, v)
    return v
```

The successor states represent the state which the current state will become when taking its legal actions. It is generated by getNextState with the current state.
We want to find the max value, so we initialize v with -inf. So that we can easy replace the value when need.
We want the highest score, so v will be replaced when the score of next legal action is higher than that of this action and return it.
When it is at the first depth. Because we want to keep the best action the pacman in game should take, we would record the best action so far into bestActions, the gloabal list variable.
The pacman will enter the max state and want to find the max score in this iteration because it want get the largest score. So it compares all the scores determined by all the successor states and the next agent (1st ghost), and pick the biggest one to return. To implement, we will assign the biggest score between v and the score of the score got from the next legal actions.
If current score v is bigger than beta, means its parents node doesn’t need to consider other states with its next legal actions because they can’t change the result and they can be pruned. The reason is that its ancestor want the min value.
Alpha represents the max score so far, so it will keep updating when we meet other higher score.



```
def min_value(state, idx_agent, alpha, beta, depth):
    v = float('inf')
    legalMoves = state.getLegalActions(idx_agent)
    for action in legalMoves:
        successor = state.getNextState(idx_agent, action)
        if depth == self.depth and idx_agent == state.getNumAgents() - 1:
            v = min(v, self.evaluationFunction(successor))
        else:
            v = min(v, value(successor, idx_agent + 1, alpha, beta, depth))
        if v < alpha:
            return v
        beta = min(beta, v)
    return v
```

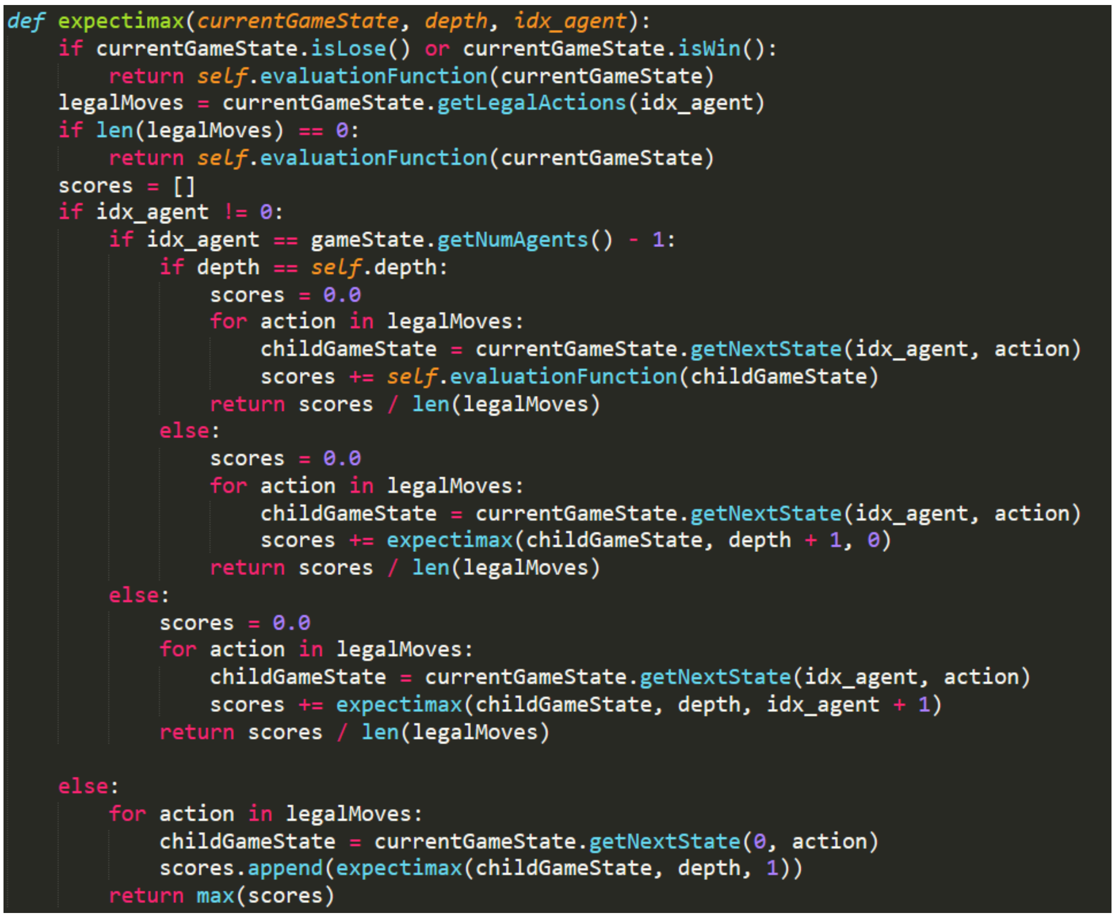
The successor states represent the state which the current state will become when taking its legal actions. It is generated by getNextState with the current state.
We want to find the min value, so we initialize v with inf. So that we can easy replace the value when need.
The ghost will enter the min state and want to find the min score in this iteration because it want to worsen the pacman. so v will be replaced when the score of next legal action is lower than that of this action and return it.
If the idx_agent equals (getNumAgents() - 1), it means it is the final ghost to discuss in this depth. Thus, it should go to the next depth and consider the pacman again. However, it should be noticed that if it is also at the bottom depth. If so, it is the parents of the leaf and it should consider the immediate score of its child states intead of going to the next depth or considering the pacman.
To implement, we will assign the smallest score between v and the score of the score got from the next legal actions.
If current score v is smaller than alpha, means its parents node doesn’t need to consider other states with its next legal actions because they can’t change the result and they can be pruned. The reason is that its ancestor want the max value.
Beta represents the min score so far, so it will keep updating when we meet other lower score.



```
bestScore = value(gameState, 0, float('-inf'), float('inf'), 1)
return random.choice(bestActions)
```

We call the function, and it will tell us the highest score. In addition, the global list variable will reserve the best actions.
Finally, we find the best action which will lead the pacman to the largest score and choose one of them to return.

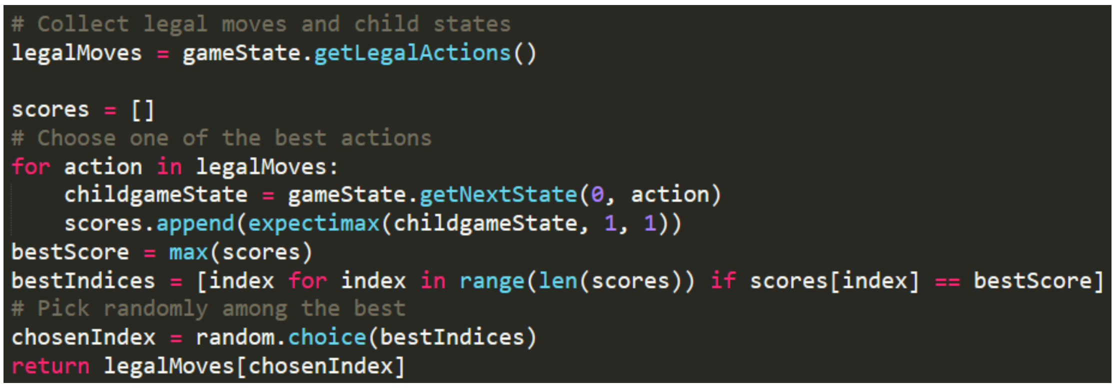
Part 3


```python
def expectimax(currentGameState, depth, idx_agent):
    if currentGameState.isLose() or currentGameState.isWin():
        return self.evaluationFunction(currentGameState)
    legalMoves = currentGameState.getLegalActions(idx_agent)
    if len(legalMoves) == 0:
        return self.evaluationFunction(currentGameState)
    scores = []
    if idx_agent != 0:
        if idx_agent == gameState.getNumAgents() - 1:
            if depth == self.depth:
                scores = 0.0
                for action in legalMoves:
                    childGameState = currentGameState.getNextState(idx_agent, action)
                    scores += self.evaluationFunction(childGameState)
                return scores / len(legalMoves)
            else:
                scores = 0.0
                for action in legalMoves:
                    childGameState = currentGameState.getNextState(idx_agent, action)
                    scores += expectimax(childGameState, depth + 1, 0)
                return scores / len(legalMoves)
        else:
            scores = 0.0
            for action in legalMoves:
                childGameState = currentGameState.getNextState(idx_agent, action)
                scores += expectimax(childGameState, depth, idx_agent + 1)
            return scores / len(legalMoves)

    else:
        for action in legalMoves:
            childGameState = currentGameState.getNextState(0, action)
            scores.append(expectimax(childGameState, depth, 1))
        return max(scores)
```

expectimax is the recursive function which will return the biggest score and help for choosing the action which pacman should take, and it assumes the average condition that all the agent will uniform randomly act their behavior to deal with non deterministic system.
Value of a state: The best achievable outcome (utility) from that state.
If isLose() or isWin() is True, it represents that there is no any child state and it is a terminal state. Thus, it should return the score of the state.
If the length of legalMoves is zero, it means that there isn’t any legal action. So, there is no any child state and it is a terminal state. Therefore, it should return the score of the state.
The list scores will record the list of scores in this iteration, and we want to choose the max or average of that accoding to the condition.
idx_agent is 0 when the agent is the pacman, or idx_agent is more than 0 when the agent is a ghost.
The child states represent the state which the current state will become when taking its legal actions. It is generated by getNextState with the current state.

The ghost will enter the chance state and want to find the average score in this iteration because we want to get expect value. So it appends all the scores determined by all the child states and the next agent into the list scores, and calcualte the average to return. Each probability of the state assumes uniformly.
If the idx_agent equals (getNumAgents() - 1), it means it is the final ghost to discuss in this depth. Thus, it should go to the next depth and consider the pacman again. However, it should be noticed that if it is also at the bottom depth. If so, it is the parents of the leaf and it should consider the immediate score of its child states intead of going to the next depth or considering the pacman.
The pacman will enter the max state and want to find the max score in this iteration because it want get the largest score. So it appends all the scores determined by all the child states and the next agent (1$^{st}$ ghost) into the list scores, and pick the biggest one to return.

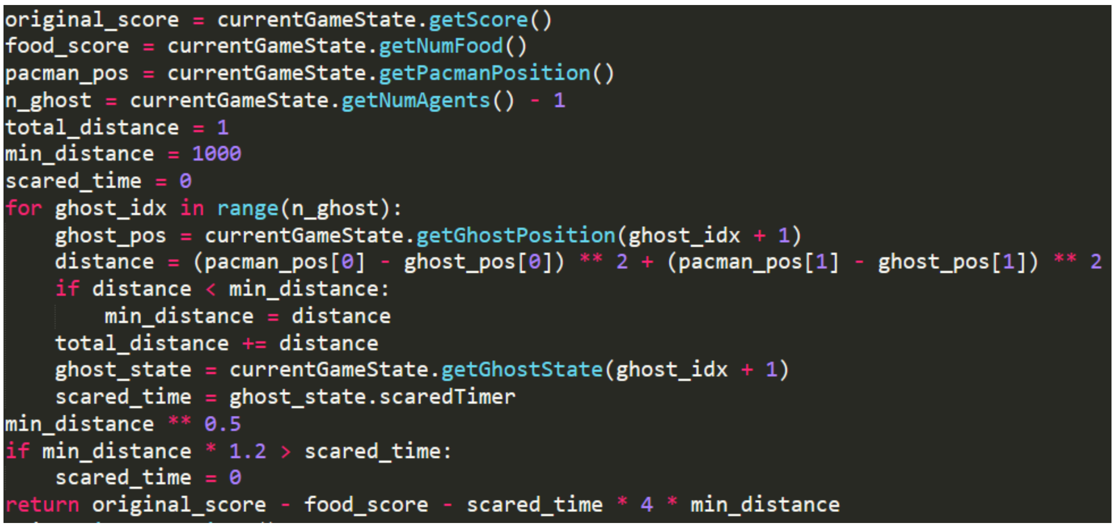


```python
# Collect legal moves and child states
legalMoves = gameState.getLegalActions()

scores = []
# Choose one of the best actions
for action in legalMoves:
    childgameState = gameState.getNextState(0, action)
    scores.append(expectimax(childgameState, 1, 1))
bestScore = max(scores)
bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
# Pick randomly among the best
chosenIndex = random.choice(bestIndices)
return legalMoves[chosenIndex]
```

First, we get the all legal actions of the state.
The list scores will record the list of scores determined by the state taking different legal actions.
We collect the different scores when the state take different lagal actions.
Next, calculate the max score and find the position of the action which will lead to the biggest score.
Finally, we find the best action which will lead the pacman to the largest score and choose one of them to return.

Part 4



```python
original_score = currentGameState.getScore()
food_score = currentGameState.getNumFood()
pacman_pos = currentGameState.getPacmanPosition()
n_ghost = currentGameState.getNumAgents() - 1
total_distance = 1
min_distance = 1000
scared_time = 0
for ghost_idx in range(n_ghost):
    ghost_pos = currentGameState.getGhostPosition(ghost_idx + 1)
    distance = (pacman_pos[0] - ghost_pos[0]) ** 2 + (pacman_pos[1] - ghost_pos[1]) ** 2
    if distance < min_distance:
        min_distance = distance
    total_distance += distance
    ghost_state = currentGameState.getGhostState(ghost_idx + 1)
    scared_time = ghost_state.scaredTimer
min_distance ** 0.5
if min_distance * 1.2 > scared_time:
    scared_time = 0
return original_score - food_score - scared_time * 4 * min_distance
```

original_score: the score we use before. The higher the score is, the better the state is.

food_score: the number of the rest of food. The the score it is, the better the state is.

We want the position of the pacman and the ghosts to calculate distance.

total_distance means the distance between the pacman and all the ghosts, and its initialize with 1 to avoid some bad situations.

min_distance represents the nearest distance between the pacman and all the ghosts, and its initialize with 1000 to easy to be replaced.

scare_time means the temporal duration that the pacman can eat the ghost.

My strategy is that if the pacman can kill the ghost, it should go forward to the ghosts to eat them for higher score. However, the time may run out when chasing the ghosts. To prevent the situation, it will effect the score when the min_distance need slightly bigger than the scare_time.

Furthermore, we should eat food ASAP in commom condition, so we will get the higher score when the food decreases.

For priority, I decide to increase the weight of eating ghost, so I multiply the min_distance by 4. The reason is that it is the most important things to eat the ghost, and it also explained why decreasing the min_distance between the pacman and the ghost will raise the score.

Finally, I combine the original score of the game before and my innovative score for better comparison.

Describe problems you meet and how you solve them.

- How to expand the node in the tree?
    - Create a recursive function.
- How to implement the assignment?
    - Understanding the core of the homework and try to understand the important function in GameState in pacman.py.
- AttributeError: 'MultiagentTreeState' object has no attribute 'getPacmanNextState'?
    - autograder.py use MultiagentTreeState instead of GameState in pacman.py, so I have to change getPacmanNextState(action) to getNextState(0, action).
- How about the score of terminal state?
    - Remember to call the evaluateFunction instead of defining by yourself.
- How to keep the optimal action in Alpha-Beta Pruning?
    - Using gloabal list variable
- Why can I pass most of the test case in part 2, but only fail the latest test case?
    - The initial value should set much bigger or smaller, instead of just setting 100 or -100.
- How to desgn new score?
    - I check the function in GameState in pacman.py, so that I can try many different strategies with available function.