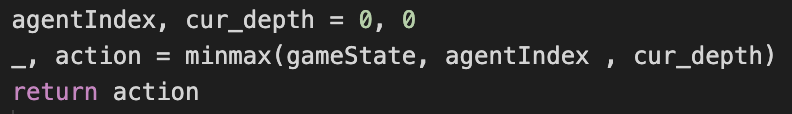
Part 1 – MinimaxAgent -- Description

1. Defined a recursive function minimax() in getAction().
2. getAction() will call minimax(state, agentIndex=0, cur\_depth=0)



1. In minimax() :

check if gameState is terminal states or not

by isWin() or sLose() or (depth == self.depth).

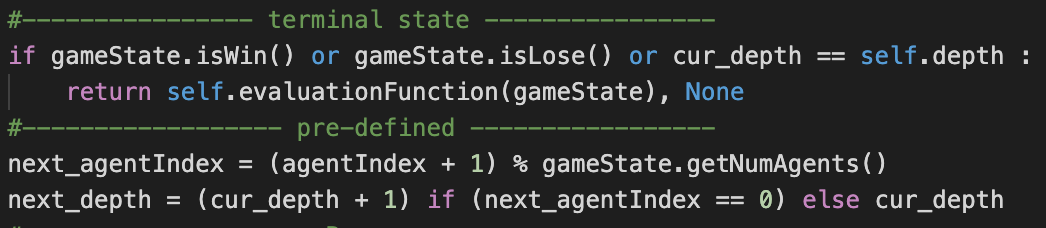
------------------------------------------------------------------------------------------------

next\_agentIndex = (agentIndex +1 ) % num(agents), which is ranging

from 0 to num(agents)-1 since Pacman has agentIndex = 0 and Ghosts

have agentIndex > 0.

next\_depth is increased based on rounds.

next\_depth +=1 at Pacman’s turns.

------------------------------------------------------------------------------------------------

If agentIndex == 0 (Pacman’s turn) :

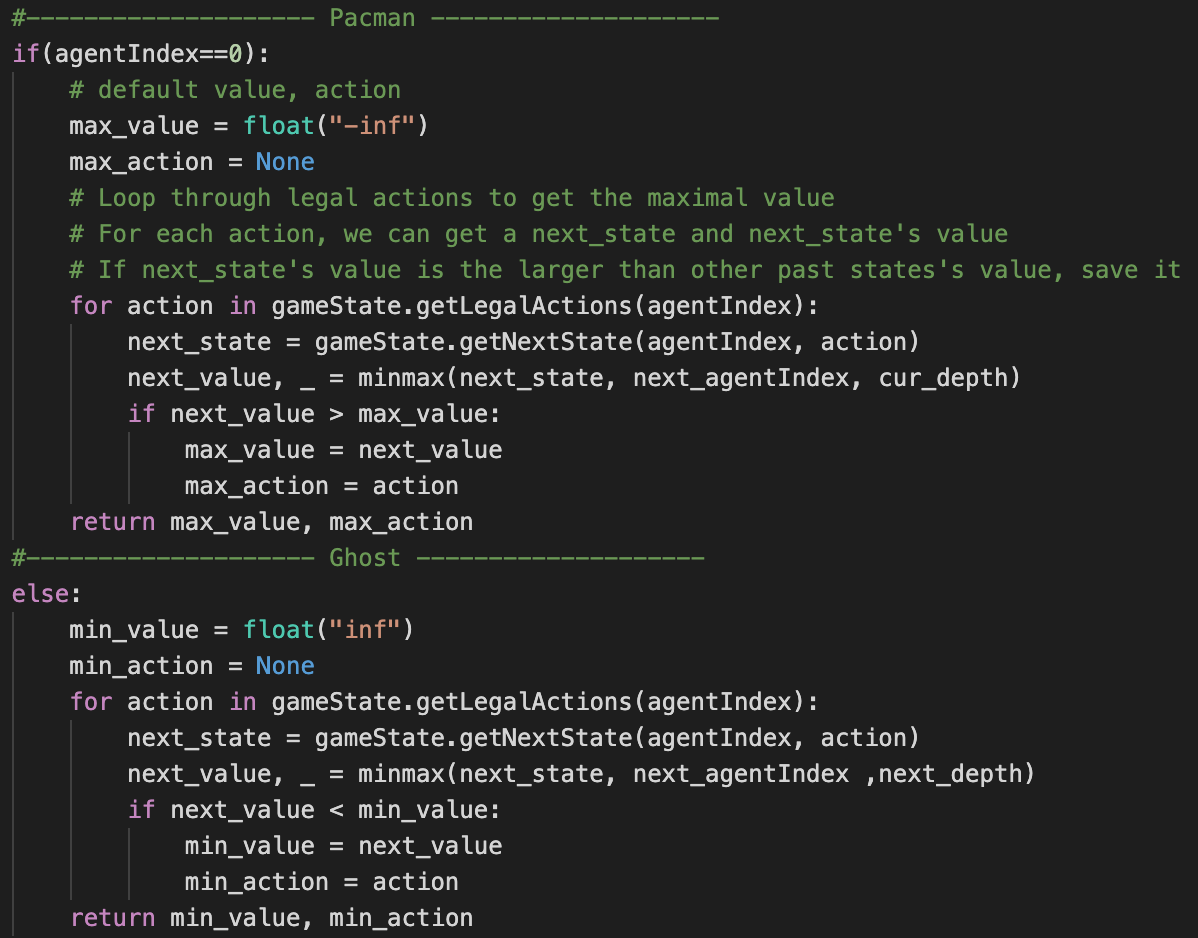
Loop through all legal actions and return (max\_value, max\_action)

Else (Ghost’s turn) :

Loop through all legal actions and return (min\_value, min\_action)

What we do for each action in the above loops is

getting next\_state by gameState.getNextState() and then getting next\_state’s value by calling minimax().



Part 1 – MinimaxAgent -- Code

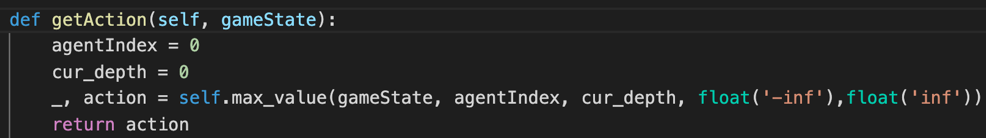


Part2 – AlphaBetaAgent -- Description

1. There are 3 functions in class AlphaBataAgent. They are getAction(),

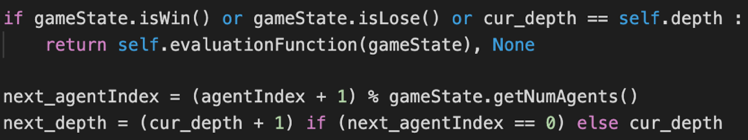
max\_value(), and min\_value().

1. getAction() call max\_value(state, index=0, depth=0, alpha=-inf, Beta=inf)



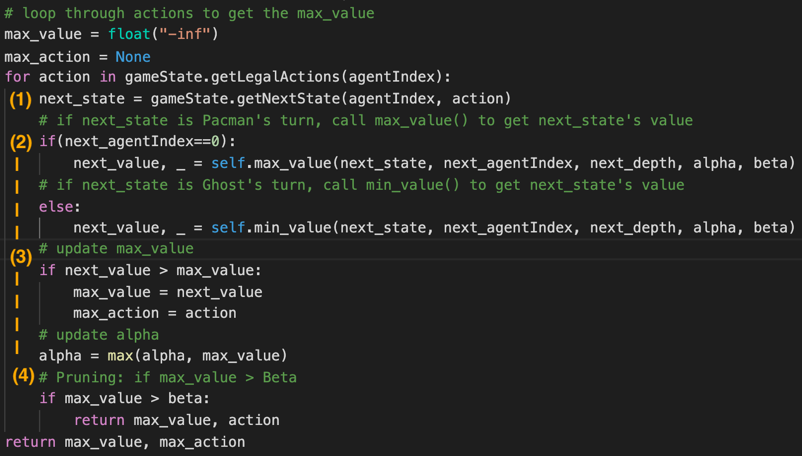
1. In max\_value() :

Check terminal states.

next\_agentIndex and next\_depth are mentioned in Part0.

------------------------------------------------------------------------------------------------

Loop through all legal actions of current state. For each action:

1. get next\_state by calling getNextState()
2. get next\_state’s value by calling max\_value() if next\_agentIndex == 0 ; otherwise call min\_value().
3. After next\_state’s value being returned, update max\_value, alpha.
4. If (max\_value > Beta) : return immediately to prune the state tree.
5. In min\_value() :

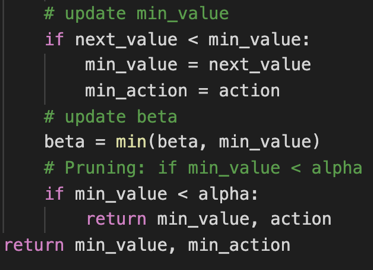
steps are same as max\_value().

Loop through all legal actions of current state. For each action:

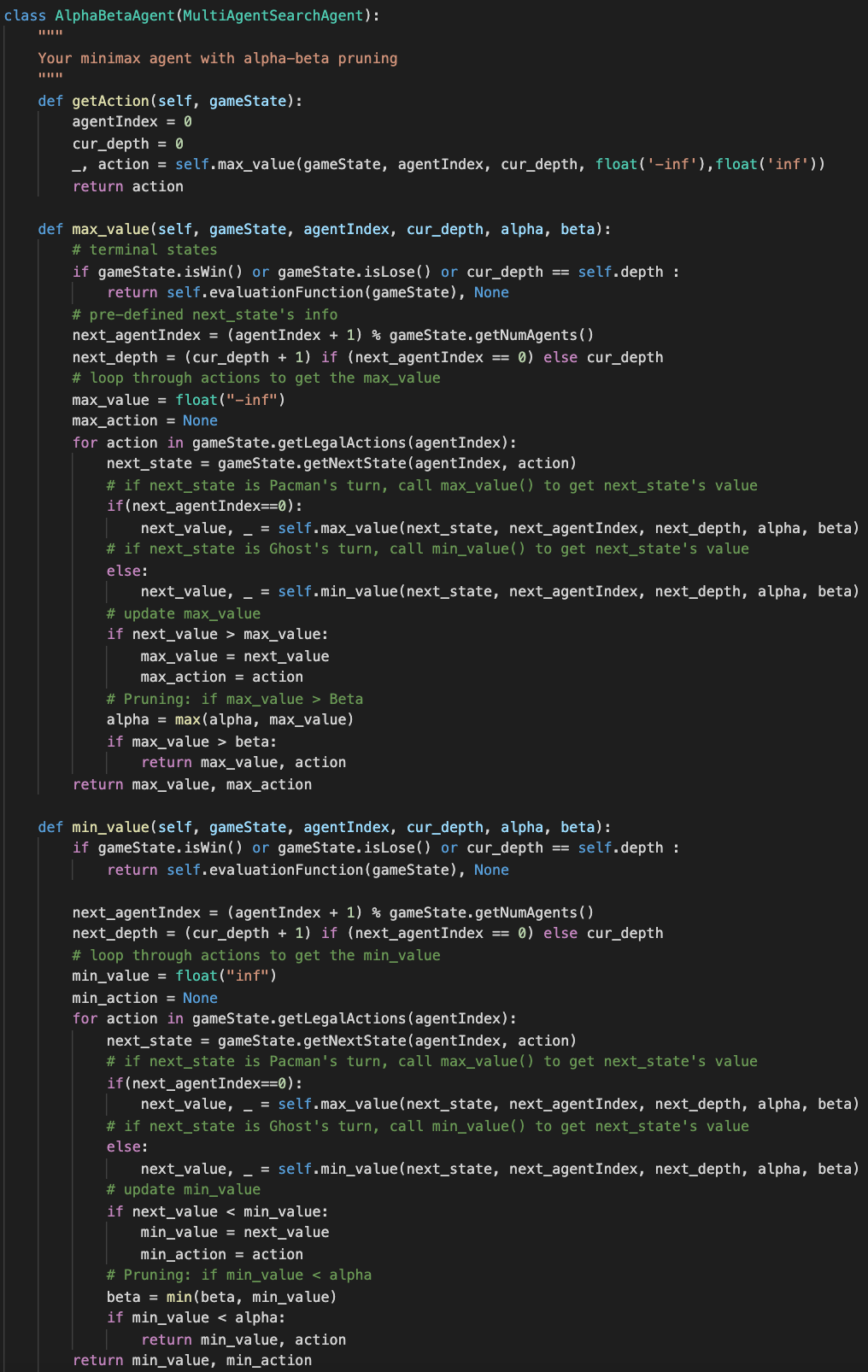
(1)(2) are same as max\_value().

(3) After next\_state’s value being returned, update min\_value, beta.

(4) If (min\_value < alpha) : return immediately to prune the state tree.

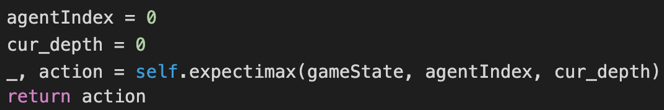


Part 2 – AlphaBetaAgent -- Code



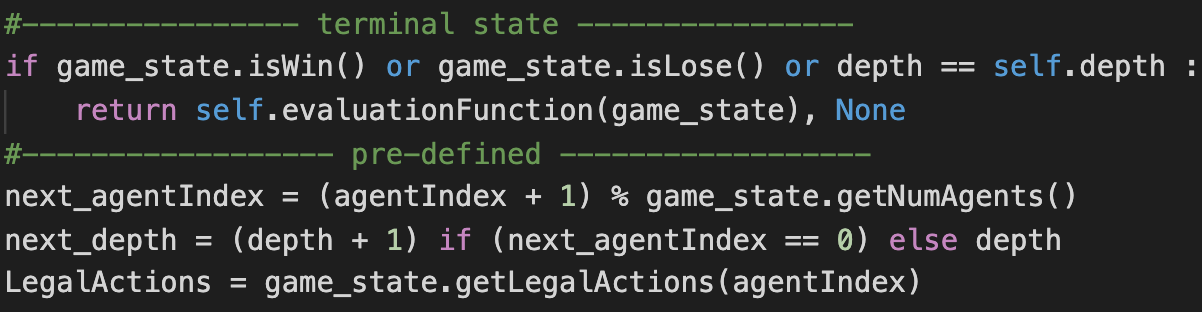
Part3 – ExpectimaxAgent -- Description

1. Define a recursive function expectimax() in getAction().
2. getAction() will call expectimax(state, agentIndex=0, cur\_depth=0)



1. In expectimax() :

check terminal states, predefine next\_agentIndex and next\_depth as

 MinimaxAgent and AlphaBeataAgent.

------------------------------------------------------------------------------------------------

If agentIndex == 0 (Pacman’s turn) :

Loop through all legal actions and return (max\_value, max\_action)

Else (Ghost’s turn) :

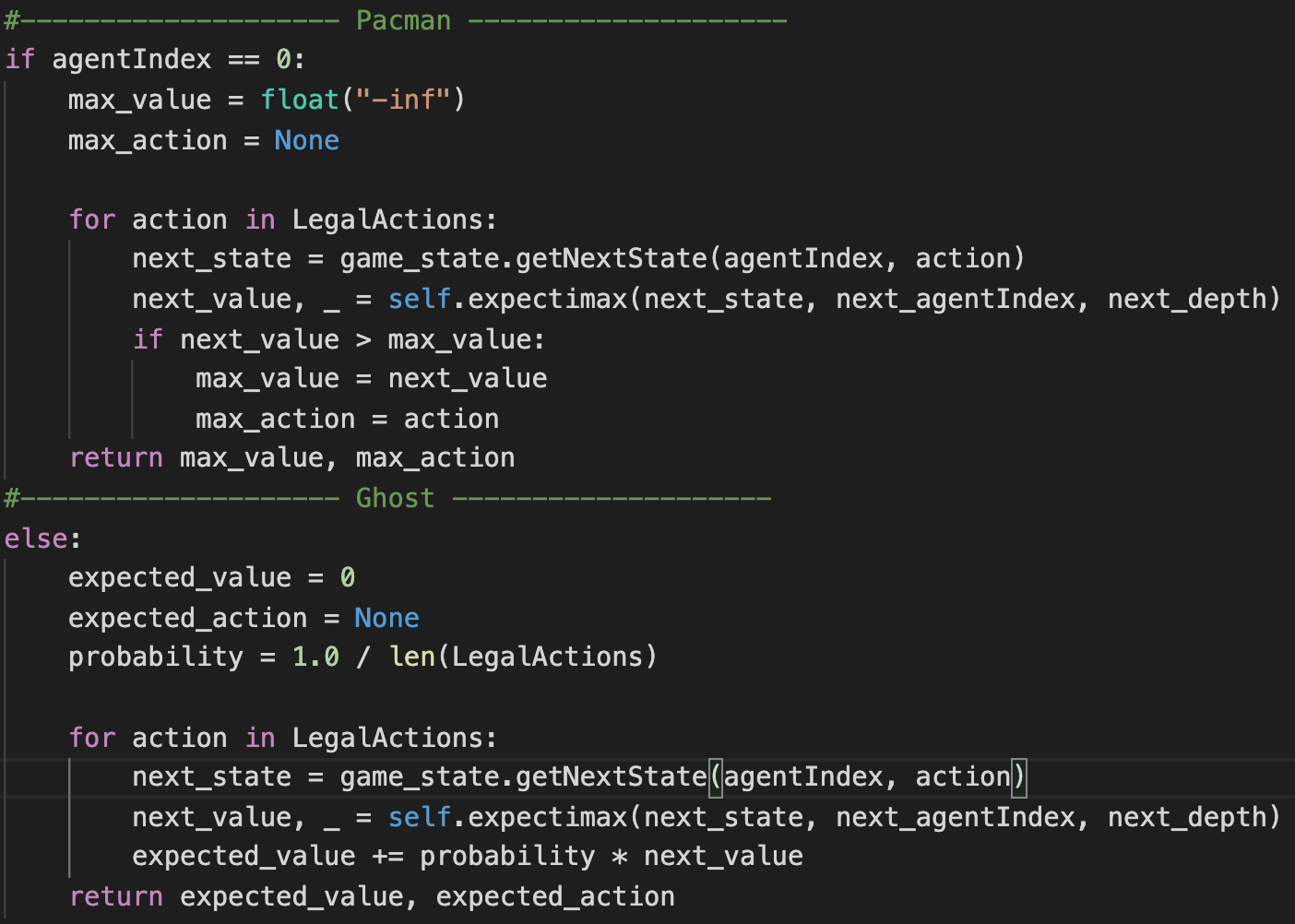
Loop through all legal actions and return (expected\_value, expected \_action)

The loop in Pacman’s part is same as what applied in MinimaxAgent’s.

The loop in Ghost’s part return (expected\_value, expected \_action),

where expected value = with p =

1/num(next\_states).



Part3 – ExpectimaxAgent -- Code

Part4 – betterEvaluationFunction -- Description & Code

1. Assume 2 kinds of factors : min\_food\_dist and game\_score
2. factor1 -- min\_food\_dist :

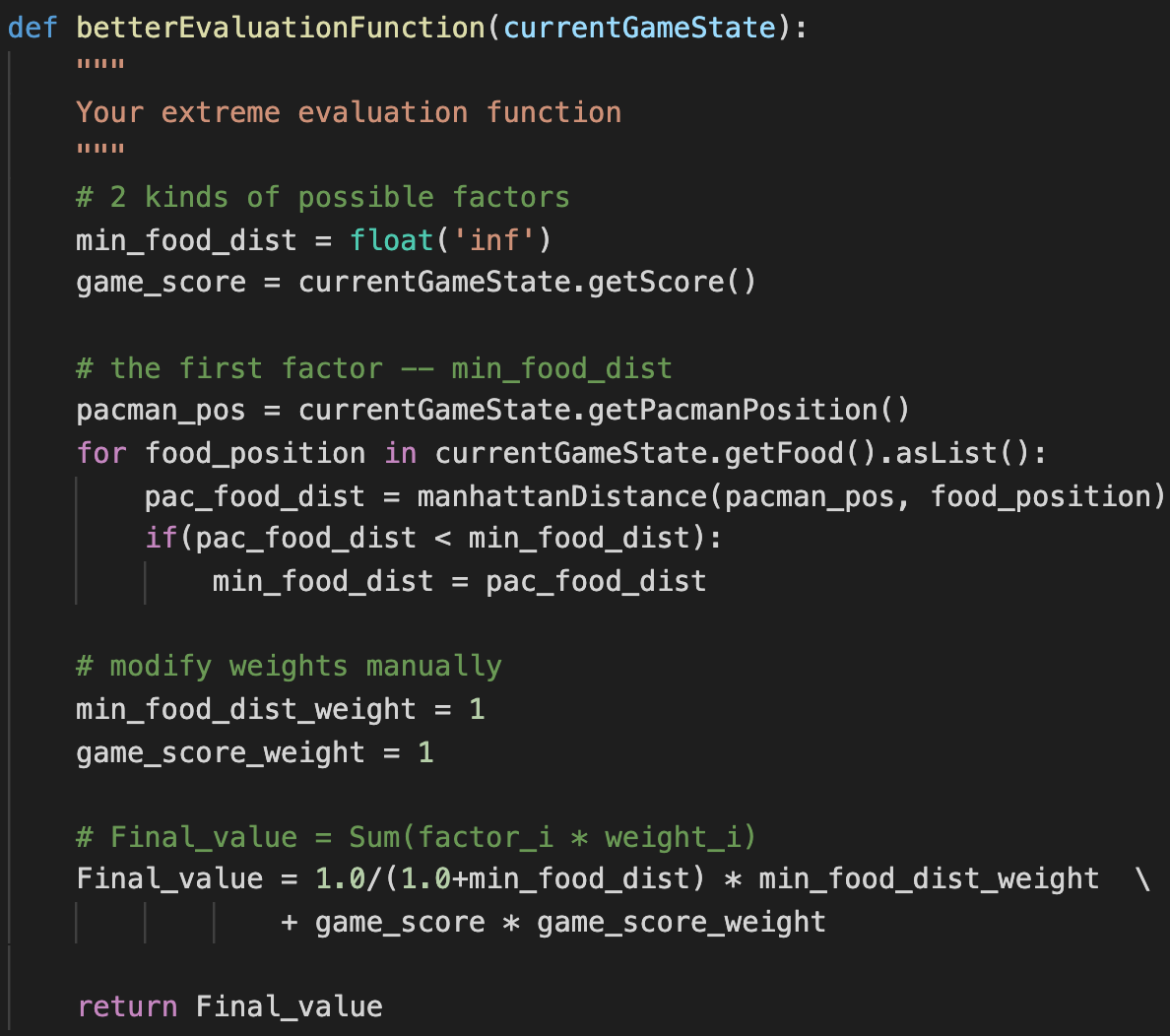
look for the minimal distance from pacman to food.

pacman’s position is rechieved by getPacmanPosition(), and

food’s position are rechieved by getFood().

The function used for caculateing distance between Pacman and Food

is manhattanDistance(), which is imported at default.

1. factor2 --game\_score is rechieved by getScore()
2. The weight for factors is set by try and error.
3. Since smaller min\_food\_dist is better, factor1 is set by 1.0/(1.0+min\_food\_dist). Add min\_food\_dist by 1 to avoid division by zero.
4. The final evaluated value = factor1\*weight1 + factor2\*weight2.

Part5 – problems you meet and how you solve them

The biggest problem is the definition of provided function.

Though there’s a short usage description at the top of MinimaxAgent code, what objects those function will return are still confusing. I believe we should go to pacman.py and utils.py frequently to look for the answers. By the way, the type hint is very helpful in exploring default function and class members(X

~~Praise for type hint~~