«باسمه تعالی»

درس مبانی و کاربردهای هوش مصنوعی

گزارش پرژه دوم

طراحی و تدوین:

مهدی رحمانی

۹۷۳۱۷۰۱

بخش اول

عامل عکس العمل

کد نوشته شده

در فایل multiAgents.py در کلاس ReflexAgent تابع evaluationFunction را به صورت زیر کامل میکنیم:

```
53
54      def evaluationFunction(self, currentGameState, action):
55          """
56          Design a better evaluation function here.
57
58          The evaluation function takes in the current and proposed successor
59          GameStates (pacman.py) and returns a number, where higher numbers are better.
60
61          The code below extracts some useful information from the state, like the
62          remaining food (newFood) and Pacman position after moving (newPos).
63          newScaredTimes holds the number of moves that each ghost will remain
64          scared because of Pacman having eaten a power pellet.
65
66          Print out these variables to see what you're getting, then combine them
67          to create a masterful evaluation function.
68          """
69          # Useful information you can extract from a GameState (pacman.py)
70          successorGameState = currentGameState.generatePacmanSuccessor(action)
71          newPos = successorGameState.getPacmanPosition()
72          newFood = successorGameState.getFood()
73          newGhostStates = successorGameState.getGhostStates()
74          newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
75          "*** YOUR CODE HERE ***"
76          # first we check the ghost scared timer is 0 or not.
77          # if the scared timer is 0 then we calculate our manhattan distance to ghost
78          # if the ghost is too close then the evaluation function should return minimum value that possible
79          # then because of this bad value pacman dont go to this state
80          # if the scared timer is not 0 then we can get closer to ghosts and it doesnt matter
81          # so here first we find manhattan distance betwwen pacman and positions of ghost states
82          ghostDistances = []
83          for ghost_state in newGhostStates:
84              if ghost_state.scaredTimer == 0:
85                  ghostDistances.append(util.manhattanDistance(newPos, ghost_state.getPosition()))
86          # if the distance < 2 ( or =0 || =1) exists in ghostDistances it means the ghost may eat us
87          if (0 in ghostDistances) or (1 in ghostDistances):
88              return -float('inf')
89
90          # we know if we eat food we can get score
91          # first we try find the closest food manhattan distance
92          closest_food_dist = float("inf")
93          newFood_list = newFood.asList()
94          for food in newFood_list:
95              closest_food_dist = min(closest_food_dist, manhattanDistance(newPos, food))
96
97          # its better for us to choose shorter distance, so the return value has an inverse relationship with each other
98          # so we try to increase the score and eat food but if the food closer, we return higher value
99          value = (1.0/closest_food_dist) + successorGameState.getScore()
100         return value
101
```

توضیح کد:

همانطور که در کامنت ها هم توضیح داده شده، در اینجا ابتدا می آییم فاصله تا روح ها را پیدا میکنیم و اگر در استیتی برویم که فاصله ما تا روح ها ۱ یا ۰ باشد خوب نیست چرا که اگر ۰ باشد ما به سراغ روح ها رفتیم و اگر ۱ باشد در گام بعدی ممکن است روح ها به سراغ ما بیایند و ببازیم. پس اگر همچین فواصلی تا روح ها برای ما پیش آمد باید تابع به ما یک مقدار بسیار کم برای مثال در اینجا منفی بینهایت برگرداند تا آن استیت انتخاب نشود. البته لازم به ذکر است در این بین چک میکنیم که اگر روح ما در حالت سفید باشد و تایمر آن ۰ نباشد میتوان بهش نزدیک شد و لازم نیست در این حالت منفی بینهایت برگرداند.

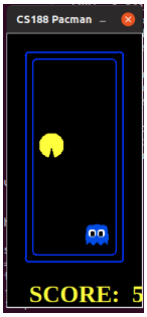
همچنین ما با خوردن غذاها امتیاز میگیریم پس اگر سعی کنیم فاصله مان با غذا را کم کنیم و آن را بخوریم برای ما خوب است. پس فاصله منهتن تا نزدیک ترین غذا را ملاک قرار میدهیم و هرچه این فاصله کمتر باشد بهتر است و باید evaluation function مقدار بیشتری برگرداند. پس رابطه عکس دارد و ما مقدار $\frac{1}{\text{فاصله تا نزدیک‌ترین غذا}}$ را میخواهیم. همچنین حالت ثانویه‌ای خوب است که امتیاز بیشتری بگیریم پس میتوان به کمک تابع getScore() امتیاز مربوطه را هم گرفت و با $\frac{1}{\text{فاصله تا نزدیک‌ترین غذا}}$ جمع کرد و به عنوان مقدار این تابع برگرداند تا براساس این مقدار تصمیم گیری پکمن انجام شود.

بنابراین در اینجا به پارامترهای فاصله از روح‌ها، فاصله تا نزدیک ترین غذا و همچنین زمان سفید بودن روح ها(به صورت غیر مستقیم) در تعیین مقدار تابع دقت شده .

امتحان عامل با دستورات گفته شده:

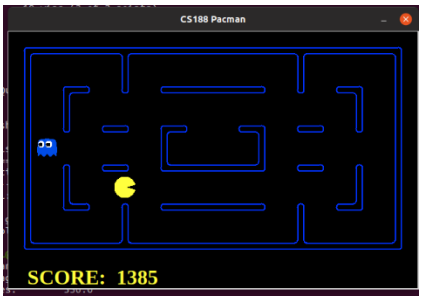
python3 pacman.py -p ReflexAgent -l testClassic

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 556
Average Score: 556.0
Scores:        556.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 564
Average Score: 564.0
Scores:        564.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 562
Average Score: 562.0
Scores:        562.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 564
Average Score: 564.0
Scores:        564.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$
```



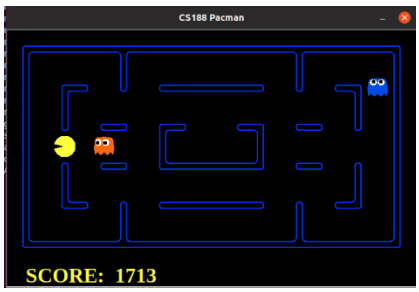
python3 pacman.py --frameTime 0 -p ReflexAgent -k 1

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 1385
Average Score: 1385.0
Scores:        1385.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$
```



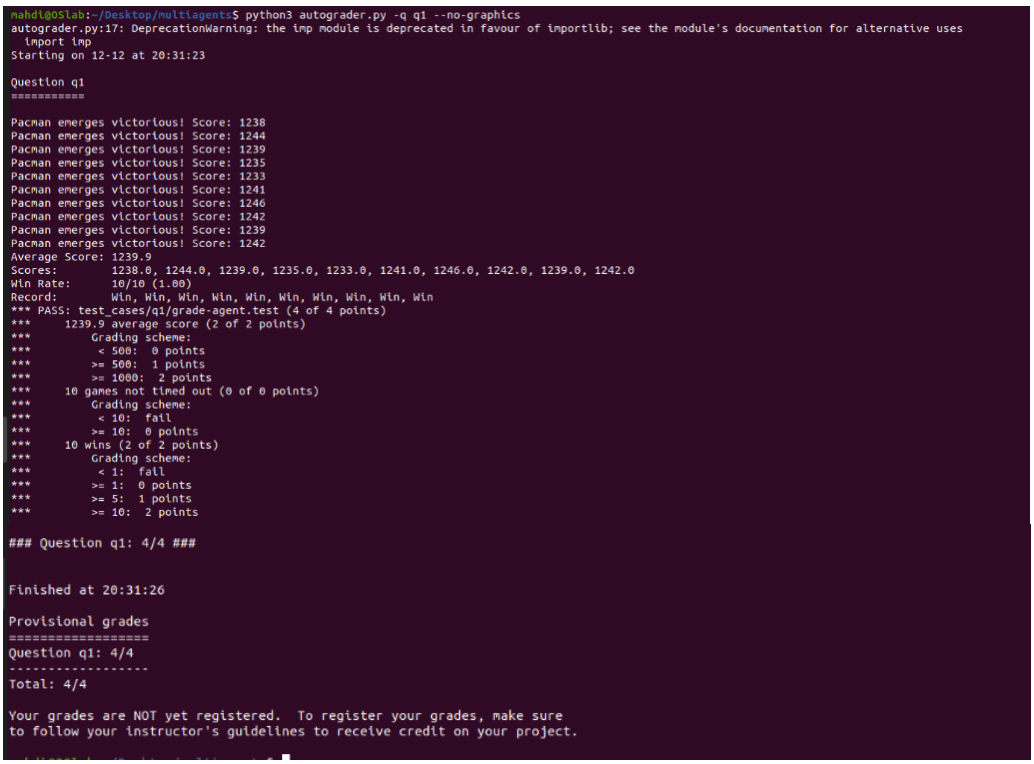
python3 pacman.py --frameTime 0 -p ReflexAgent -k 2

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py --frameTime 0 -p ReflexAgent -k 2
Pacman emerges victorious! Score: 1713
Average Score: 1713.0
Scores:        1713.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$
```



ارزیابی سوال اول:

python3 autograder.py -q q1 --no-graphics



بخش دوم

مینیماکس

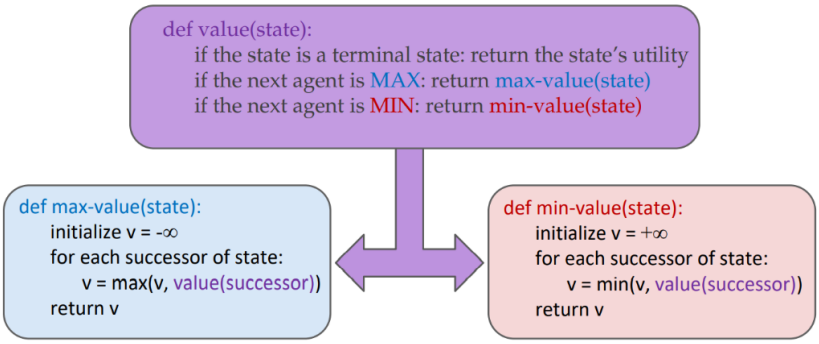
کد نوشته شده

در فایل multiAgents.py در کلاس MinimaxAgent تابع getAction را به صورت زیر کامل میکنیم:

```
    def getAction(self, gameState):
        """
        Returns the minimax action from the current gameState using self.depth
        and self.evaluationFunction.

        Here are some method calls that might be useful when implementing minimax.

        gameState.getLegalActions(agentIndex):
        Returns a list of legal actions for an agent
        agentIndex=0 means Pacman, ghosts are >= 1

        gameState.generateSuccessor(agentIndex, action):
        Returns the successor game state after an agent takes an action

        gameState.getNumAgents():
        Returns the total number of agents in the game

        gameState.isWin():
        Returns whether or not the game state is a winning state

        gameState.isLose():
        Returns whether or not the game state is a losing state
        """
        "*** YOUR CODE HERE ***"
        # we want to choose best action
        # first we should find possible successors for this state of pacman
        # then we should find best action to best successor for get maximum value
        # for do this job we need find the value for possible succesors with the help of value function(minimax algorithm)
        # then we choose maximum value from them and return the action related to that
        value_action = (-float("inf"), None)
        for action in gameState.getLegalActions(self.index):
            new_value_action = ((self.value(gameState.generateSuccessor(self.index, action), 1, 1)), action)
            value_action = max(value_action, new_value_action, key=lambda x:x[0])
        return value_action[1]
    # according to slides our pseudocode for minimax function:
    """
        def value(state):
            if the state is a terminal state: return the state's utility
            if the next agent is MAX: return max_value(state)
            if the next agent is MIN: return min_value(state)
    """
    def value(self, gameState, agentIndex, depth):
        # first step: we check if the state is a terminal state or not
        if gameState.isWin() or gameState.isLose() or depth == self.depth * gameState.getNumAgents():
            return self.evaluationFunction(gameState)
        # second step: if the agentIndex == 0 so next agent is MAX
        if agentIndex == 0:
            return self.max_value(gameState, agentIndex, depth)
        # third step: if the agentIndex>0 so next agent is MIN
        else:
            return self.min_value(gameState, agentIndex, depth)


    # according to slides our pseudocode for max_value function:
    """
        def max_value(state):
            initialize v = - inf
            for each succesor of state:
                v = max(v, value(successor))
            return v
    """
    def max_value(self, gameState, agentIndex, depth):
        # first step : initialize v = - inf
        v = -float("inf")
        # second step : find successors list
        legal_actions = gameState.getLegalActions(agentIndex)
        successor_list = []
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
            successor_list.append(successor)
        # third step : find maximum value and update v
        for successor in successor_list:
            v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
        return v

    # according to slides our pseudocode for min_value function:
    """
        def min_value(state):
            initialize v = + inf
            for each succesor of state:
                v = min(v, value(successor))
            return v
    """
    def min_value(self, gameState, agentIndex, depth):
        # first step : initialize v = + inf
        v = float("inf")
        # second step : find successors list
        legal_actions = gameState.getLegalActions(agentIndex)
        successor_list = []
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
            successor_list.append(successor)
        # third step : find minimum value and update v
        for successor in successor_list:
            v = min(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
        return v
```

توضیح کد:

در این قسمت با توجه به شبه کدی که در اسلایدها ی درس داشتیم تابع را پیاده سازی کردیم:



یک تابع به عنوانvalue نیاز داریم که ۳ تا حالت مختلف دارد:

۱) در حالت ترمینال اگر باشیم مقدار utility آن استیت را به کمک evaluationFunction حساب میکنیم و برمیگردانیم.

۲) اگر agent بعدی گره ماکس باشد(که این را به کمک agentIndex تشخیص میدهیم) آنگاه به کمک تابع max_value مقدار v را محاسبه میکنیم. به این صورت که ابتدا مقدار اولیه v منفی بینهایت گرفته میشود و سپس ابتدا legal_action ها را میبابیم و بعد به کمک آن ها لیست successor ها را میابیم و روی آن حلقه میزنیم و بعد تابع value را روی successorها فراخوانی میکنیم و بین v آن ها و v که داشتیم ماکسیمم میگیریم. این فراخوانی ها به صورت بازگشتی ادامه میابند تا به حالتی که به ترمینال استیت ها هست برسیم.

۳) اگر agent بعدی min باشد که در این صورت باید agentIndex مخالف ۰ باشد، مانند گام دو میباشد میباشد با این تفاوت که باید بین v ها مینیمم بگیریم. در ضمن v اولیه هم مثبت بینهایت است.

در رابطه با ترمینال استیت باید گفت که زمانی که به استیتی برسیم که برنده یا بازنده باشد یا که در برگ های درخت باشد درواقع به ترمینال استیت رسیدیم. برای اینکه بفهیم به عمق درخت رسیدیم یا خیر به این ترتیب عمل کردیم:

میدانیم در هر عمق یک بار پکمن بازی میکند و یک بار هر کدام از روح ها. میدانیم که هربار که agent روح داشته باشیم یک سطح min داریم و هربار agent پکمن داشته باشیم یک سطح ماکس داریم پس میتوان دو کار کرد یکی اینکه به ازای هربار صدا زدن max_value به متغیر عمق یک واحد زیاد کنیم یا اینکه عمق را کلا برابر با self.depth نگیریم بلکه برابر با self.depth * number of agents بگیریم و بعد هم به ازای صدا زدن

min_value و هم max_value یه واحد به متغیر عمق اضافه شود. درواقع ابتدا پکمن در عمق ۰ است و بازی با آن شروع میشود و هربار ۱ واحد به متغیر depth اضافه میشود و شرط رسیدن به عمق نیز این میباشد:

```
depth ==  self.depth * gameState.getNumAgents()
```

حال طریقه حساب کردن بهترین action به کمک مینیماکس:

میدانیم شروع کار با پکمن میباشد و باید تصمیم بگیرد action انتخابی آن چی باشد بهتر است.

برای این کار گام اول که پکمن هست را خودمان دستی حسابی میکنیم و در گام بعد که ابتدا نوبت روح ها هست به کمک تابع value مقدار v را برای هرکدام از successorsهای به دست آمده از legalAction ها حساب میکنیم. بعد مقدار v و همچنینaction متناظر با آن را به صورت یک دوتایی نگه میداریم و بین vها ماکسیمم را انتخاب میکنیم و action متناظر با آن را به عنوان خروجی برمیگردانیم.

امتحان عامل با دستورات گفته شده:

python3 pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$
```

python3 pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0
Win Rate:      0/1 (0.00)
Record:        Loss
mahdi@OSlab:~/Desktop/multiagents$
```

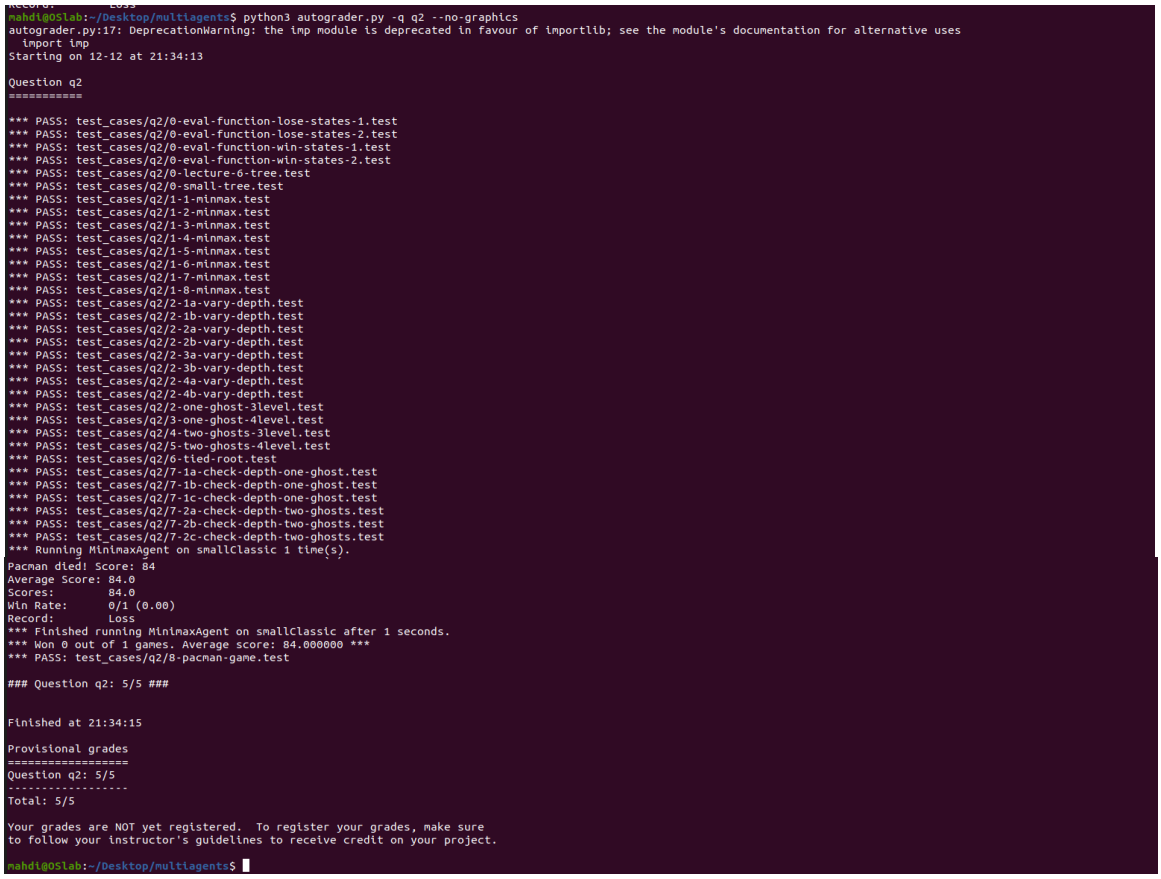
بررسی کنید

چرا پکمن در این حالت به دنبال باخت سریع تر است؟

در این جا باتوجه به اینکه از minimax استفاده کردیم هم روح ها میخواهند ما ببازیم هم ما میخواهیم ببریم. درواقع به بیانی روح ها تخاصمی بازی میکنند. لذا در این جا شرایطی پیش می آید که هر دو روح به سمت پکمن می آیند و پکمن هم راه فراری ندارد و محاصره میشود. پکمن هم زمانی که نوبت تصمیم گیریش هست چون گره max میباشد به دنبال بیشینه کردن utility میباشد پس سعی میکند تا سریع تر خودکشی کند تا زمان کمتری بگذرد و امتیاز کمتری از دست بدهد و درواقع بیشینه امتیازی که میتواند بگیرد این است که خودکشی کند و راهی برای جمع امتیاز ندارد.

ارزیابی سوال دوم:

python3 autograder.py -q q2 --no-graphics



بخش سوم

هرس آلفا-بتا

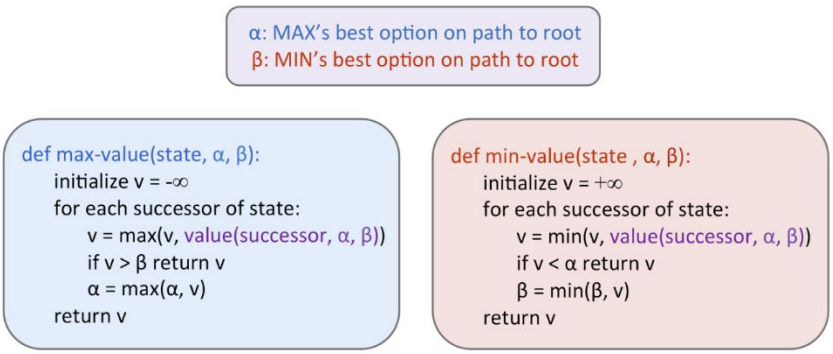
کد نوشته شده

در فایل multiAgents.py در کلاس AlphaBetaAgent تابع getAction را به صورت زیر کامل میکنیم:

```python
    def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"
        # we want to choose best action
        # first we should find possible successors for this state of pacman
        # then we should find best action to best successor for get maximum value
        # for do this job we need find the value for possible succesors with the help of value function(alpha_beta algorithm)
        # then we choose maximum value from them and return the action related to that
        value_action = (-float("inf"), None)
        a, b = -float("inf"), float("inf")
        for action in gameState.getLegalActions(self.index):
            new_value_action = ((self.value(gameState.generateSuccessor(self.index, action), 1, 1, a, b)), action)
            value_action = max(value_action, new_value_action, key=lambda x:x[0])
            # the condition below never be true because in first step b = +inf
            # if v>b return v
            a = max(a, value_action[0])
        return value_action[1]

    # according to slides our pseudocode for alpha_beta function:
    """
        def value(state, a, b):
            if the state is a terminal state: return the state's utility
            if the next agent is MAX: return max_value(state)
            if the next agent is MIN: return min_value(state)
    """
    def value(self, gameState, agentIndex, depth, a, b):
        # first step: we check if the state is a terminal state or not
        if gameState.isWin() or gameState.isLose() or depth == self.depth * gameState.getNumAgents():
            return self.evaluationFunction(gameState)
        # second step: if the agentIndex == 0 so next agent is MAX
        if agentIndex == 0:
            return self.max_value(gameState, agentIndex, depth, a, b)
        # third step: if the agentIndex>0 so next agent is MIN
        else:
            return self.min_value(gameState, agentIndex, depth, a, b)


    # according to slides our pseudocode for max_value function:
    """
        def max_value(state, a, b):
            initialize v = - inf
            for each succesor of state:
                v = max(v, value(successor, a, b))
                if v > b return v
                a = max(a, v)
            return v
    """
    def max_value(self, gameState, agentIndex, depth, a, b):
        # first step : initialize v = - inf
        v = -float("inf")
        # second step : find each successor(hear we want prunning so we shouldn't find complete list of successors like q2)
        legal_actions = gameState.getLegalActions(agentIndex)
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
        # third step : find maximum value and update v
            v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1, a, b)))
            if v > b :
                return v
            a = max(a, v)
        return v

    # according to slides our pseudocode for min_value function:
    """
        def min_value(state, a, b):
            initialize v = + inf
            for each succesor of state:
                v = min(v, value(successor, a, b))
                if v < a return v
                b = min(a, v)
            return v
    """
    def min_value(self, gameState, agentIndex, depth, a, b):
        # first step : initialize v = + inf
        v = float("inf")
        # second step : find each successor(hear we want prunning so we shouldn't find complete list of successors like q2)
        legal_actions = gameState.getLegalActions(agentIndex)
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
        # third step : find minimum value and update v
            v = min(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1, a, b)))
            if v < a :
                return v
            b = min(b, v)
        return v
```

توضیح کد:

برای پیاده سازی این قسمت از شبه کد داخل اسلایدها و همچنین نکته ای که در دستور کار گفته شده(که تساوی را برای هرس قرار ندهیم) استفاده شده است:



کد این سوال خیلی مشابه سوال ۲ میباشد منتها نیاز به برخی تغییرات دارد.

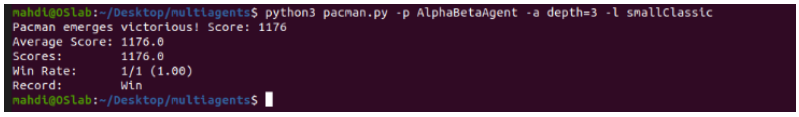
اولا در ورودی توابع باید آلفا و بتا (که در کد a و b) هستند را نیز قرار دهیم. مقدار اولیه آلفا منفی بینهایت و مقدار اولیه بتا مثبت بی نهایت است.

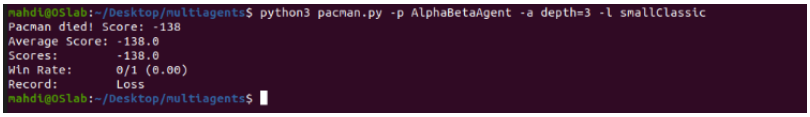
در این حالت دیگر همه successor ها بررسی نمیشوند. بنابراین به این صورت عمل شده که ابتدا legal actionها را میابیم و بعد متناظر با آن successor را میابیم و عملیات داخل حلقه را انجام میدهیم. در اینجا ممکن است که مقدار v بزرگ تر از بتا شود در max_value و یا v کوچکتر از آلفا شود در min_value و همانجا درجا مقدار v را برگردانیم و نیازی به بررسی بقیه successorها نباشد.

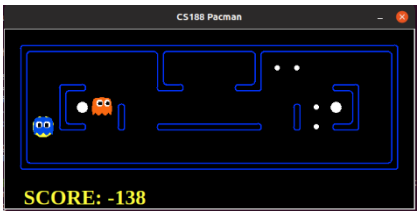
ما بقی توضیحات مانند قسمت قبل میباشد.

امتحان عامل با دستورات گفته شده:

python3 pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic



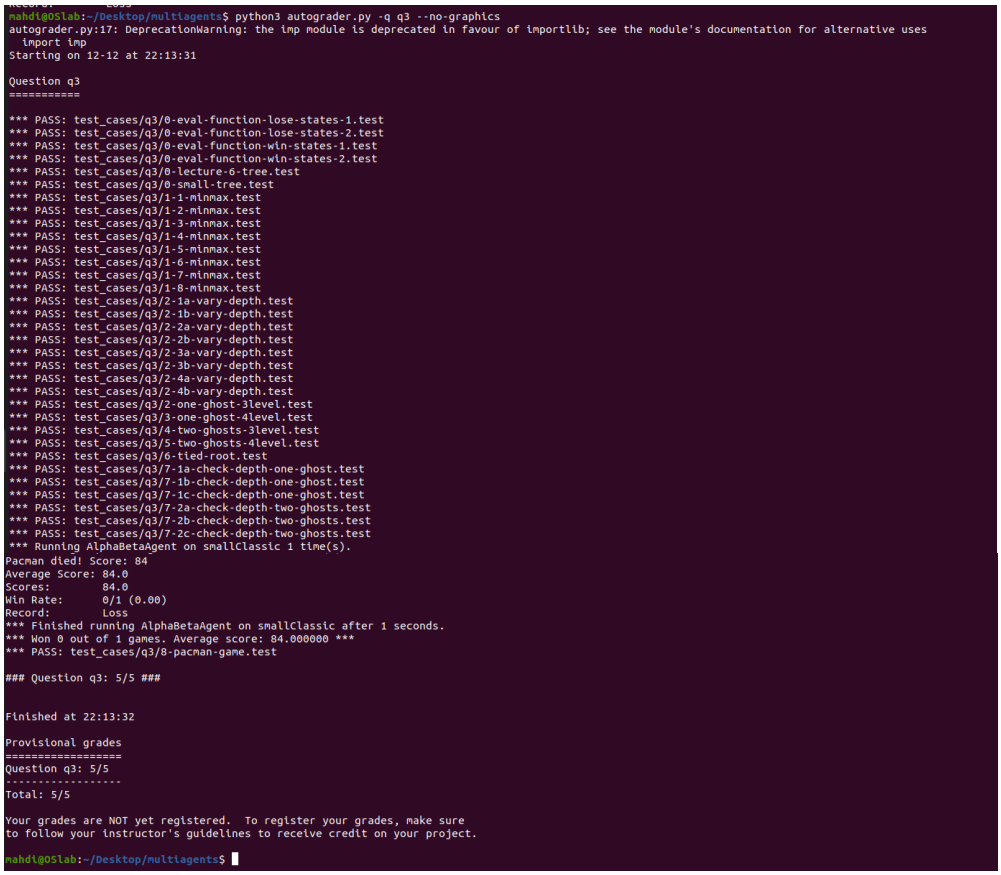




همچنین دستور را برای عمق ۲ با عامل MinimaxAgent نیز اجرا کردم و زمان نتیجه گیری تقریبا برابر بود.

ارزیابی سوال سوم:

python3 autograder.py -q q3 --no-graphics



بخش چهارم

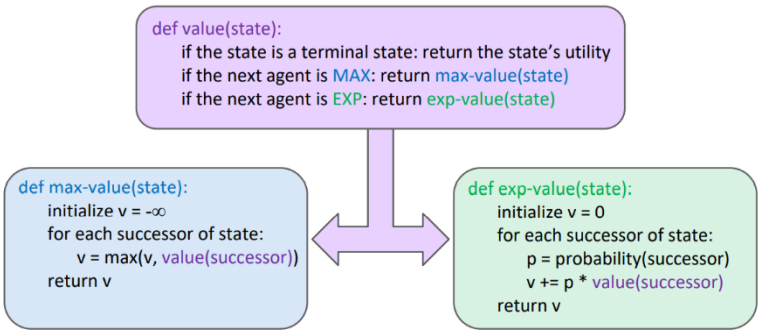
مینیماکس احتمالی

در فایل multiAgents.py در کلاس ExpectimaAgent تابع getAction را به صورت زیر کامل میکنیم:

```python
    def getAction(self, gameState):
        """
        Returns the expectimax action using self.depth and self.evaluationFunction

        All ghosts should be modeled as choosing uniformly at random from their
        legal moves.
        """
        "*** YOUR CODE HERE ***"
        # we want to choose best action
        # first we should find possible successors for this state of pacman
        # then we should find best action to best successor for get maximum value
        # for do this job we need find the value for possible succesors with the help of value function(Expectimax algorithm)
        # then we choose maximum value from them and return the action related to that
        value_action = (-float("inf"), None)
        for action in gameState.getLegalActions(self.index):
            new_value_action = ((self.value(gameState.generateSuccessor(self.index, action), 1, 1)), action)
            value_action = max(value_action, new_value_action, key=lambda x:x[0])
        return value_action[1]

    # according to slides our pseudocode for Expectimax function:
    """
        def value(state):
            if the state is a terminal state: return the state's utility
            if the next agent is MAX: return max_value(state)
            if the next agent is EXP: return exp_value(state)
    """
    def value(self, gameState, agentIndex, depth):
        # first step: we check if the state is a terminal state or not
        if gameState.isWin() or gameState.isLose() or depth == self.depth * gameState.getNumAgents():
            return self.evaluationFunction(gameState)
        # second step: if the agentIndex == 0 so next agent is MAX
        if agentIndex == 0:
            return self.max_value(gameState, agentIndex, depth)
        # third step: if the agentIndex>0 so next agent is MIN
        else:
            return self.exp_value(gameState, agentIndex, depth)


    # according to slides our pseudocode for max_value function:
    """
        def max_value(state):
            initialize v = - inf
            for each succesor of state:
                v = max(v, value(successor))
            return v
    """
    def max_value(self, gameState, agentIndex, depth):
        # first step : initialize v = - inf
        v = -float("inf")
        # second step : find successors list
        legal_actions = gameState.getLegalActions(agentIndex)
        successor_list = []
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
            successor_list.append(successor)
        # third step : find maximum value and update v
        for successor in successor_list:
            v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
        return v

    # according to slides our pseudocode for exp_value function:
    """
        def exp_value(state):
            initialize v = 0
            for each succesor of state:
                p = probability(successor)
                v += p*value(successor)
            return v
    """
    def exp_value(self, gameState, agentIndex, depth):
        # first step : initialize v = 0
        v = 0
        # second step : find successors list
        legal_actions = gameState.getLegalActions(agentIndex)
        successor_list = []
        for action in legal_actions:
            successor = gameState.generateSuccessor(agentIndex, action)
            successor_list.append(successor)
        # third step : find probability and update v
        for successor in successor_list:
            p = 1.0/len(legal_actions)
            v += p * self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)
        return v
```

توضیح کد:

در اینجا نیز طبق شبه کدی که در اسلایدهای درس میباشد پیاده سازی را انجام داده ایم:



همانطور که مشاهده میشود الگوریتم خیلی شبیه minimax میباشد. تابع max_value مشابه آنچه که در minimax پیاده سازی کردیم پیاده شده است.

در اینجا برای تابع value نیز ۳ حالت داریم.

۱) اگر حالت ترمینال باشد که مشابه بحثی که در سوال دوم برای حالت ترمینال کردیم، آنگاه با کمک evaluation function مقدار را برمیگرداند.

۲) در حالتی که agnet بعدی ما MAX باشد یعنی agentIndex ما برابر با ۰ باشد مانند مینیماکس تابع max_value صدا زده میشود و جزئیات آن نیز به همان ترتیب قبلا گفته شده است.

۳) در حالتی که agentIndex ما غیر ۰ باشد باید به سراغ تابع exp_value برویم. درواقع روح های ما دیگر مثل سابق با قطعیت رفتار نمیکنند و بلکه احتمال وارد کار میشود. درواقع ارواح ازبین حرکات مجازشان به صورت تصادفی و با احتمالی برابر یکی را انتخاب میکنند. پس در اینجا ما یک تابع exp_value داریم که در آن مقدار اولیه v برابر ۰ است و بعد برای حرکات مجاز successor متناظر را انتخاب کرده و همچنین مقدار احتمال برای آن successor را حساب کرده که در اینجا برابر با : $\frac{1}{\text{تعداد حرکات مجاز برای حرکت روح}}$ میباشد. سپس مقدار v با ضرب این احتمال در value(successor) که به صورت بازگشتی حساب میشود، جمع و آپدیت میشود.

ما بقی توضیحات ماند مینیماکس است. در این جا نیز از پکمن شروع میکنیم و در تابع getAction ابتدا successorهای آن را به دست آورده و برای هرکدام تابع value را صدا میزنیم. چون در استیتی که نوبت پکمن است و agentIndex برابر ۰ است ما نود Max هستیم پس بین valueها ماکس گرفته و action متناظر با این valueرا در خروجی getAction برمیگردانیم.

امتحان عامل با دستورات گفته شده:

python3 pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3

```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
Pacman emerges victorious! Score: 511
Average Score: 511.0
Scores:        511.0
Win Rate:      1/1 (1.00)
Record:        Win
mahdi@OSlab:~/Desktop/multiagents$
```


```
mahdi@OSlab:~/Desktop/multiagents$ python3 pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
Pacman died! Score: -494
Average Score: -494.0
Scores:        -494.0
Win Rate:      0/1 (0.00)
Record:        Loss
mahdi@OSlab:~/Desktop/multiagents$
```

ارزیابی سوال چهارم:

python3 autograder.py -q q4 --no-graphics

```
mahdi@OSlab:~/Desktop/multiagents$ python3 autograder.py -q q4 --no-graphics
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
Starting on 12-12 at 22:46:17

Question q4
==========

*** PASS: test_cases/q4/0-eval-function-lose-states-1.test
*** PASS: test_cases/q4/0-eval-function-lose-states-2.test
*** PASS: test_cases/q4/0-eval-function-win-states-1.test
*** PASS: test_cases/q4/0-eval-function-win-states-2.test
*** PASS: test_cases/q4/0-expectimax1.test
*** PASS: test_cases/q4/1-expectimax2.test
*** PASS: test_cases/q4/2-one-ghost-3level.test
*** PASS: test_cases/q4/3-one-ghost-4level.test
*** PASS: test_cases/q4/4-two-ghosts-3level.test
*** PASS: test_cases/q4/5-two-ghosts-4level.test
*** PASS: test_cases/q4/6-1a-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1b-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1c-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q4/7-pacman-game.test

### Question q4: 5/5 ###


Finished at 22:46:18

Provisional grades
==================
Question q4: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

mahdi@OSlab:~/Desktop/multiagents$
```

بررسی کنید:

روش مینیماکس در موقعیتی که در دام قرار گرفته باشد خودش اقدام به باختن و پایان سریعتر بازی میکند ولی در صورت استفاده از مینیماکس احتمالی در ۵۰ درصد از موارد برنده میشود. این سناریو را با هر دو روش امتحان کنید و درستی این گزاره را نشان دهید

ابتدا به بررسی هرس آلفا-بتا میپردازیم. برای این کار دستور زیر را میزنیم:

python3 pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10

همانطور که مشاهده میشود در تمام بازی باخته است و چون در دام قرار گرفته سعی کرده بازی را سریعتر تمام کند و ببازد.

حال به بررسی روش expectimax میپردازیم. برای این کار دستور زیر را اجرا میکنیم:

python3 pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10

همانطور که مشاهده میشود در ۷ بازی از ۱۰ بازی توانسته برنده شود. درواقع در اینجا در ۷۰ درصد از مواقع برنده بوده که طبق گفته سوال بیش از ۵۰ درصد میباشد.

پس توانستیم درستی گزاره گفته شده را نشان دهیم.

بخش پنجم

تابع ارزیابی

کد نوشته شده

در فایل multiAgents.py تابع betterEvaluationFunction() را به صورت زیر کامل میکنیم:

```python
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DESCRIPTION: <write something here so we know what you did>
    """
    "*** YOUR CODE HERE ***"
    """
      Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
      evaluation function (question 5).
      DESCRIPTION: <write something here so we know what you did>
      In this evaluation function we want to consider : 1) distance to closest ghost
                                                        2) distance to closest food
                                                        3) number of remaining capsules
                                                        4) number of remaining food
                                                        5) game score

    """
    # like question1 we should first find useful information from Gamestate
    pacman_pos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    ghost_positions = currentGameState.getGhostPositions()
    # initialize our parameters:
    #   distance to ghost
    closest_ghost_distance = 0
    #   distance to closest food
    closest_food_distance = float('inf')
    #   number of remaining capsules
    remain_capsule_num = 0
    #   number of remaining food
    remain_food_num = 0
    #   game score
    game_score = 0

    # first we check if we close to a ghost we should return a very low value
    ghost_distances = []
    for ghost_pos in ghost_positions:
        ghost_distance = manhattanDistance(pacman_pos, ghost_pos)
        ghost_distances.append(ghost_distance)
        if (ghost_distance == 0 or ghost_distance == 1):
            return -float('inf')
    if len(ghost_distances) != 0:
        closest_ghost_distance = min(ghost_distances)

    # second we want to calculate our distance to closest food
    for food_pos in newFood.asList():
        closest_food_distance = min(closest_food_distance, manhattanDistance(pacman_pos, food_pos))

    # third we want to calculate number of remaining capsules
    # we plus 1 because of avoiding to be 0 and getting divide by zero error
    remain_capsule_num = len(currentGameState.getCapsules()) + 1

    # fourth we want to calculate number of remaining food
    # we plus 1 because of avoiding to be 0 and getting divide by zero error
    remain_food_num = currentGameState.getNumFood() + 1

    # fifth we want to calculate the score of game in this state
    game_score = currentGameState.getScore()

    # a vector that contains the parametes:
    parameters = [closest_ghost_distance, 1.0 / closest_food_distance, 1.0 / remain_capsule_num, 1.0 / remain_food_num, game_score]

    # a vector that contains the weights related to each parameter
    weights = [1, 1000, 10000, 1000, 100]

    # calculating evaluation value
    value = 0
    for parameter, weight in zip(parameters, weights):
        value += parameter * weight

    return value
```

توضیح کد:

این قسمت نیز تا حدودی شبیه سوال ۱ میباشد منتها برای اینکه دقیق تر و بهتر نتیجه بگیریم تعداد پارامترهای موثر در مقداری که تابع بر میگرداند را افزایش دادیم و همچنین برای آن‌ها وزن درنظر گرفتیم.

در اینجا پارامترهایی که من در نظر گرفتم به این ترتیب هستند:

۱) فاصله تا نزدیک ترین روح
۲) فاصله تا نزدیک ترین غذا
۳) تعداد کپسول‌های باقی مانده
۴) تعداد غذاهای باقی مانده
۵) امتیاز بازی در این استیت

در ابتدای کد یک سری متغیر برای نگهداری این مقادیر گفته شده تعریف شده است. ابتدا به سراغ بررسی فاصله تا روح ها رفتیم.

چنانچه فاصله از روحی ۰ یا ۱ باشد باید یک مقدار خیلی کم برای مثال در اینجا منفی بینهایت برگردانیم تا این استیت انتخاب نشود. چراکه اگر فاصله ۰ باشد یعنی روح ها به ما برخورد کردند و اگر ۱ هم باشد در استپ بعد که نوبت روح است میتواند به سمت ما بیاید و باز ببازیم.

در این حین فاصله منهتن تا روح ها را در یک لیست ذخیره میکنیم و فاصله تا نزدیک ترین روح را با مینیمم گیری در آن لیست میابیم.

سپس به سراغ حساب کردن فاصله منهتن تا غذاها میرویم و فاصله تا نزدیک ترین غذا را نیز نگه میداریم.

به کمک تابع هایی از gameState میتوانیم تعداد غذاهای باقی مانده و کپسول های باقی مانده را بیابیم. لازم است به مقدار آن ها یک واحد اضافه کنیم تا بعدا در صورتی که غذا یا کپسولی نمانده بود ارور تقسیم بر ۰ نگیریم.

حال که پارامترها را به دست آوردیم باید به رابطه آن نیز توجه کنیم و بر این اساس آن ها را در یک لیست قرار دهیم. میدانیم فاصله تا روح ها هرچه بیشتر باشد بهتر است پس مقدار نهایی با فاصله از نزدیک ترین روح رابطه مستقیم دارد. میدانیم فاصله تا غذا ها هرچه کم باشد بهتر است پس مقدار نهایی با فاصله از نزدیک ترین غذا رابطه عکس دارد. میدانیم هرچه تعداد کپسول های باقی مانده و تعداد غذاهای باقی مانده کمتر شود بهتر است پس با این موارد نیز رابطه عکس داریم. هرچه game_score ما بالاتر باشد بهتر است پس با آن رابطه مستقیم داریم.

حال به سراغ لیست وزن های متناظر با موارد گفته شده میرویم:

میدانیم که فاصله از روح ها برای ما مهم است منتها خطرناک ترین حالت ها زمانی بوده است که فاصله ما با آن ها ۱ یا ۰ بوده که درجا مقدار منفی بینهایت برگرداندیم. پس در سایر موارد حاشیه امن داریم و اولویت بقیه موارد میتواند بیشتر باشد. پس به همین خاطر به آن وزن ۱ دادم.

میزان نزدیک بودن به غذاها و تعداد غذای باقی مانده تقریبا به یک اندازه اهمیت دارند و من به هردو وزن ۱۰۰۰ دادم. منتها کپسول ها چون برای ما منفعت بیشتری دارند پس به آن وزن بیشتری دادم که برابر ۱۰۰۰۰ است. البته این ۱۰۰۰ و۱۰۰۰۰ چون تقسیم برتعداد باقی مانده ها یا فاصله میشوند مقدارشان کمتر و معتدل تر میشود.

سپس به game_score نیز ضریب ۱۰۰ دادم. منتها چون تقسیم بر چیزی نمیشود اثرش بیشتر است. زیرا ما هم هدفمان بیشتر کردن امتیاز است و این خیلی کمک کننده است.

*** لازم به ذکر است که با سعی و خطا به این مقادیر رسیدم. ابتدا دلایلی مثل بالا در ذهن داشتم و کمی اعداد را بالا و پایین کردم تا به یک نتیجه متعادل و خوب رسیدم.

اگر ضریب game score کم و در حد ۱۰ شود ارور میخوریم و بازی به درستی انجام نمیشود. اگر مقدار ضریب مربوط به تعداد غذاهای مانده و فاصله از کمترین غذا را کم کنیم مثلا بذاریم یک میانگین score ما میشود خدودا ۶۰۰ و این بد است. این دو چون کمی در راستای یک دیگرند کم و زیاد شدن جفتشان باهم تاثیر خود را بیشتر نشان میدهد.

اگر مقدار ضریب کپسول ها را هم کم کنیم score پایین می آید و اگر در حد ۱ بذاریم حتی بعضا خطا میگیریم.

در نهایت ضرایب را در مقادیر حساب شده متناظر ضرب کرده و همه را با هم جمع میکنیم و آن را به عنوان مقدار نهایی برمیگردانیم.

ارزیابی سوال پنجم:

python3 autograder.py -q q5 --no-graphics