

«باسمه تعالی»



درس مبانی و کاربردهای هوش مصنوعی

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

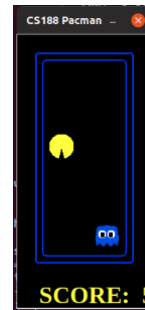
همچنین ما با خوردن غذاها امتیاز میگیریم پس اگر سعی کنیم فاصله مان با غذا را کم کنیم و آن را بخوریم برای ما خوب است. پس فاصله منتهن تا نزدیک ترین غذا را ملاک قرار میدهم و هرچه این فاصله کمتر باشد بهتر است و باید evaluation function مقدار بیشتری برگرداند. پس رابطه عکس دارد و ما مقدار $\frac{1}{\text{فاصله تا نزدیک ترین غذا}}$ را میخواهیم. همچنین حالت ثانویه ای خوب است که امتیاز بیشتری بگیریم پس میتوان به کمک تابع `getScore()` امتیاز مربوطه را هم گرفت و با $\frac{1}{\text{فاصله تا نزدیک ترین غذا}}$ 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 روح ها، 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:

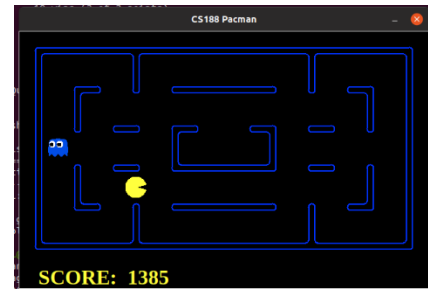
python3 pacman.py -p ReflexAgent -l testClassic

```
nahdi@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
nahdi@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
nahdi@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
nahdi@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
nahdi@OSLab:~/Desktop/multiagents$
```



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

```
nahdi@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
nahdi@OSLab:~/Desktop/multiagents$
```



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

```
nahdi@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
nahdi@OSLab:~/Desktop/multiagents$
```



ارزیابی سوال اول:

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

```
nahdi@oslab:~/Desktop/multiagents$ python3 autograder.py -q q1 --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 20:31:23

Question q1
=====

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1235
Pacman emerges victorious! Score: 1233
Pacman emerges victorious! Score: 1241
Pacman emerges victorious! Score: 1246
Pacman emerges victorious! Score: 1242
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1242
Average Score: 1239.9
Scores:      1238.0, 1244.0, 1239.0, 1235.0, 1233.0, 1241.0, 1246.0, 1242.0, 1239.0, 1242.0
Win Rate:    10/10 (1.00)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q1/grade-agent.test (4 of 4 points)
*** 1239.9 average score (2 of 2 points)
***   Grading scheme:
***     < 500: 0 points
***     >= 500: 1 points
***     >= 1000: 2 points
*** 10 games not timed out (0 of 0 points)
***   Grading scheme:
***     < 10: fail
***     >= 10: 0 points
*** 10 wins (2 of 2 points)
***   Grading scheme:
***     < 1: fail
***     >= 1: 0 points
***     >= 5: 1 points
***     >= 10: 2 points

### Question q1: 4/4 ###

Finished at 20:31:26

Provisional grades
=====
Question q1: 4/4
-----
Total: 4/4

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

بخش دوم

مینیماکس

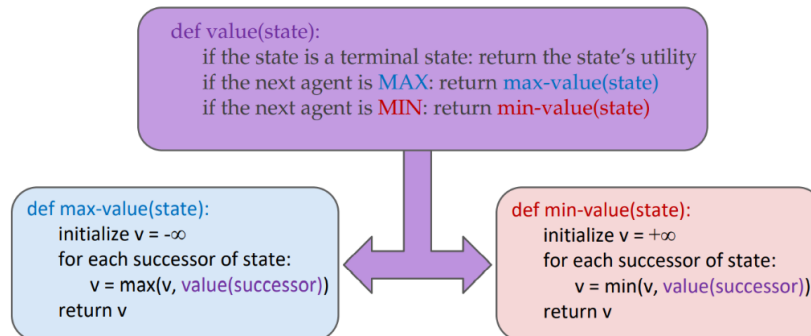
کد نوشته شده

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

```
343 def getAction(self, gameState):
344     """
345     Returns the minimax action from the current gameState using self.depth
346     and self.evaluationFunction.
347
348     Here are some method calls that might be useful when implementing minimax.
349
350     gameState.getLegalActions(agentIndex):
351     Returns a list of legal actions for an agent
352     agentIndex=0 means Pacman, ghosts are >= 1
353
354     gameState.generateSuccessor(agentIndex, action):
355     Returns the successor game state after an agent takes an action
356
357     gameState.getNumAgents():
358     Returns the total number of agents in the game
359
360     gameState.isWin():
361     Returns whether or not the game state is a winning state
362
363     gameState.isLose():
364     Returns whether or not the game state is a losing state
365
366     """
367     """ YOUR CODE HERE """
368     # we want to choose best action
369     # first we should find possible successors for this state of pacman
370     # then we should find best action to best successor for get maximum value
371     # for do this job we need find the value for possible successors with the help of value function(minimax algorithm)
372     # then we choose maximum value from then and return the action related to that
373     value_action = (-float("inf"), None)
374     for action in gameState.getLegalActions(self.index):
375         new_value_action = (self.value(gameState.generateSuccessor(self.index, action), 1, 1), action)
376         value_action = max(value_action, new_value_action, key=lambda x:x[0])
377     return value_action[1]
378
379 # according to slides our pseudocode for minimax function:
380 """
381     def value(state):
382         if the state is a terminal state: return the state's utility
383         if the next agent is MAX: return max value(state)
384         if the next agent is MIN: return min value(state)
385 """
386 def value(self, gameState, agentIndex, depth):
387     # first step: we check if the state is a terminal state or not
388     if gameState.isWin() or gameState.isLose() or depth == self.depth * gameState.getNumAgents():
389         return self.evaluationFunction(gameState)
390     # second step: if the agentIndex == 0 so next agent is MAX
391     if agentIndex == 0:
392         return self.max_value(gameState, agentIndex, depth)
393     # third step: if the agentIndex > 0 so next agent is MIN
394     else:
395         return self.min_value(gameState, agentIndex, depth)
396
397 # according to slides our pseudocode for max value function:
398 """
399     def max_value(state):
400         initialize v = - inf
401         for each successor of state:
402             v = max(v, value(successor))
403         return v
404 """
405 def max_value(self, gameState, agentIndex, depth):
406     # first step : initialize v = - inf
407     v = -float("inf")
408     # second step : find successors list
409     legal_actions = gameState.getLegalActions(agentIndex)
410     successor_list = []
411     for action in legal_actions:
412         successor = gameState.generateSuccessor(agentIndex, action)
413         successor_list.append(successor)
414     # third step : find maximum value and update v
415     for successor in successor_list:
416         v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
417     return v
418
419 # according to slides our pseudocode for min_value function:
420 """
421     def min_value(state):
422         initialize v = + inf
423         for each successor of state:
424             v = min(v, value(successor))
425         return v
426 """
427 def min_value(self, gameState, agentIndex, depth):
428     # first step : initialize v = + inf
429     v = float("inf")
430     # second step : find successors list
431     legal_actions = gameState.getLegalActions(agentIndex)
432     successor_list = []
433     for action in legal_actions:
434         successor = gameState.generateSuccessor(agentIndex, action)
435         successor_list.append(successor)
436     # third step : find minimum value and update v
437     for successor in successor_list:
438         v = min(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
439     return v
440
```

توضیح کد:

در این قسمت با توجه به شبهه کدی که در اسلایدهای درس داشتیم تابع را پیاده سازی کردیم:



یک تابع به عنوان value 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 میابیم و بعد به کمک آن ها لیست `successor` ها را میابیم و روی آن حلقه میزنیم و بعد تابع `value` را روی `successor` ها 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 می باشد می باشد 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 بفهمیم 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 کلاً برابر با `self.depth` بگیریم بلکه برابر با `self.depth * number of agents` بگیریم و بعد هم به ازای صدا زدن

min_value و max_value به 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 هر کدام از successorهای به دست آمده از legalAction ها حساب میکنیم. بعد مقدار v و همچنین action 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

```
mahdi@OSLab:~/Desktop/multiagents$ python3 autograder.py -q q2 --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 21:34:13

Question q2
=====
*** PASS: test_cases/q2/0-eval-function-lose-states-1.test
*** PASS: test_cases/q2/0-eval-function-lose-states-2.test
*** PASS: test_cases/q2/0-eval-function-win-states-1.test
*** PASS: test_cases/q2/0-eval-function-win-states-2.test
*** PASS: test_cases/q2/0-lecture-6-tree.test
*** PASS: test_cases/q2/0-small-tree.test
*** PASS: test_cases/q2/1-1-minmax.test
*** PASS: test_cases/q2/1-2-minmax.test
*** PASS: test_cases/q2/1-3-minmax.test
*** PASS: test_cases/q2/1-4-minmax.test
*** PASS: test_cases/q2/1-5-minmax.test
*** PASS: test_cases/q2/1-6-minmax.test
*** PASS: test_cases/q2/1-7-minmax.test
*** PASS: test_cases/q2/1-8-minmax.test
*** PASS: test_cases/q2/2-1a-vary-depth.test
*** PASS: test_cases/q2/2-1b-vary-depth.test
*** PASS: test_cases/q2/2-2a-vary-depth.test
*** PASS: test_cases/q2/2-2b-vary-depth.test
*** PASS: test_cases/q2/2-3a-vary-depth.test
*** PASS: test_cases/q2/2-3b-vary-depth.test
*** PASS: test_cases/q2/2-4a-vary-depth.test
*** PASS: test_cases/q2/2-4b-vary-depth.test
*** PASS: test_cases/q2/2-one-ghost-3level.test
*** PASS: test_cases/q2/3-one-ghost-4level.test
*** PASS: test_cases/q2/4-two-ghosts-3level.test
*** PASS: test_cases/q2/5-two-ghosts-4level.test
*** PASS: test_cases/q2/6-tied-root.test
*** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2c-check-depth-two-ghosts.test
*** Running MInimaxAgent 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 MInimaxAgent on smallClassic after 1 seconds.
*** Non 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q2/8-pacman-game.test

### Question q2: 5/5 ###

Finished at 21:34:15

Provisional grades
=====
Question q2: 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$
```

بخش سوم

هرس آلفابتا

کد نوشته شده

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

```
247 def getAction(self, gameState):
248     """
249     Returns the minimax action using self.depth and self.evaluationFunction
250     """
251     """ YOUR CODE HERE """
252     # We want to choose best action
253     # first we should find possible successors for this state of pacman
254     # then we should find best action to best successor for get maximum value
255     # for do this job we need find the value for possible successors with the help of value function(alpha_beta algorithm)
256     # then we choose maximum value from them and return the action related to that
257     value_action = (-float("inf"), None)
258     a, b = -float("inf"), float("inf")
259     for action in gameState.getLegalActions(self.index):
260         new_value_action = (self.value(gameState.generateSuccessor(self.index, action), 1, 1, a, b), action)
261         value_action = max(value_action, new_value_action, key=lambda x: x[0])
262         # the condition below never be true because in first step b = +inf
263         # if v>b return v
264         a = max(a, value_action[0])
265     return value_action[1]
266
267 # according to slides our pseudocode for alpha_beta function:
268 """
269 def value(state, a, b):
270     if the state is a terminal state: return the state's utility
271     if the next agent is MAX: return max_value(state)
272     if the next agent is MIN: return min_value(state)
273 """
274 def value(self, gameState, agentIndex, depth, a, b):
275     # first step: we check if the state is a terminal state or not
276     if gameState.isWin() or gameState.isLose() or depth == self.depth + gameState.getNumAgents():
277         return self.evaluationFunction(gameState)
278     # second step: if the agentIndex == 0 so next agent is MAX
279     if agentIndex == 0:
280         return self.max_value(gameState, agentIndex, depth, a, b)
281     # third step: if the agentIndex>0 so next agent is MIN
282     else:
283         return self.min_value(gameState, agentIndex, depth, a, b)
284
285 # according to slides our pseudocode for max_value function:
286 """
287 def max_value(state, a, b):
288     initialize v = - inf
289     for each successor of state:
290         v = max(v, value(successor, a, b))
291         if v > b return v
292     a = max(a, v)
293     return v
294 """
295 def max_value(self, gameState, agentIndex, depth, a, b):
296     # first step : initialize v = - inf
297     v = -float("inf")
298     # second step : find each successor (here we want pruning so we shouldn't find complete list of successors like q2)
299     legal_actions = gameState.getLegalActions(agentIndex)
300     for action in legal_actions:
301         successor = gameState.generateSuccessor(agentIndex, action)
302         # third step : find maximum value and update v
303         v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1, a, b)))
304         if v > b :
305             return v
306         a = max(a, v)
307     return v
308
309 # according to slides our pseudocode for min_value function:
310 """
311 def min_value(state, a, b):
312     initialize v = + inf
313     for each successor of state:
314         v = min(v, value(successor, a, b))
315         if v < a return v
316         b = min(b, v)
317     return v
318 """
319 def min_value(self, gameState, agentIndex, depth, a, b):
320     # first step : initialize v = + inf
321     v = float("inf")
322     # second step : find each successor (here we want pruning so we shouldn't find complete list of successors like q2)
323     legal_actions = gameState.getLegalActions(agentIndex)
324     for action in legal_actions:
325         successor = gameState.generateSuccessor(agentIndex, action)
326         # third step : find minimum value and update v
327         v = min(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1, a, b)))
328         if v < a :
329             return v
330         b = min(b, v)
331     return v
332
```

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:

α : MAX's best option on path to root
 β : MIN's best option on path to root

```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = -\infty$   
    for each successor of state:  
         $v = \max(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v > \beta$  return  $v$   
         $\alpha = \max(\alpha, v)$   
    return  $v$ 
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = +\infty$   
    for each successor of state:  
         $v = \min(v, \text{value}(\text{successor}, \alpha, \beta))$   
        if  $v < \alpha$  return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

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

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

در این حالت دیگر همه successor ها بررسی نمیشوند. بنابراین به این صورت عمل شده که ابتدا legal action ها 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 β شود در \max_value و یا v کوچکتر از α شود در \min_value و همانجا درجا مقدار v را برگردانیم و نیازی به بررسی بقیه successor ها نباشد.

ما بقی توضیحات مانند قسمت قبل میباشد.

امتحان عامل با دستورات گفته شده:

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

```
mahdi@05Lab:~/Desktop/multiagents$ python3 pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman emerges victorious! Score: 1176
Average Score: 1176.0
Scores: 1176.0
Win Rate: 1/1 (1.00)
Record: Win
mahdi@05Lab:~/Desktop/multiagents$
```



```
mahdi@05Lab:~/Desktop/multiagents$ python3 pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
Pacman died! Score: -138
Average Score: -138.0
Scores: -138.0
Win Rate: 0/1 (0.00)
Record: Loss
mahdi@05Lab:~/Desktop/multiagents$
```



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

ارزیابی سوال سوم:

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

```
mahdi@051ab:~/Desktop/multiagents$ python3 autograder.py -q q3 --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 22:13:31

Question q3
=====

*** PASS: test_cases/q3/0-eval-function-lose-states-1.test
*** PASS: test_cases/q3/0-eval-function-lose-states-2.test
*** PASS: test_cases/q3/0-eval-function-win-states-1.test
*** PASS: test_cases/q3/0-eval-function-win-states-2.test
*** PASS: test_cases/q3/0-lecture-6-tree.test
*** PASS: test_cases/q3/0-small-tree.test
*** PASS: test_cases/q3/1-1-minmax.test
*** PASS: test_cases/q3/1-2-minmax.test
*** PASS: test_cases/q3/1-3-minmax.test
*** PASS: test_cases/q3/1-4-minmax.test
*** PASS: test_cases/q3/1-5-minmax.test
*** PASS: test_cases/q3/1-6-minmax.test
*** PASS: test_cases/q3/1-7-minmax.test
*** PASS: test_cases/q3/1-8-minmax.test
*** PASS: test_cases/q3/2-1a-vary-depth.test
*** PASS: test_cases/q3/2-1b-vary-depth.test
*** PASS: test_cases/q3/2-2a-vary-depth.test
*** PASS: test_cases/q3/2-2b-vary-depth.test
*** PASS: test_cases/q3/2-3a-vary-depth.test
*** PASS: test_cases/q3/2-3b-vary-depth.test
*** PASS: test_cases/q3/2-4a-vary-depth.test
*** PASS: test_cases/q3/2-4b-vary-depth.test
*** PASS: test_cases/q3/2-one-ghost-3level.test
*** PASS: test_cases/q3/3-one-ghost-4level.test
*** PASS: test_cases/q3/4-two-ghosts-3level.test
*** PASS: test_cases/q3/5-two-ghosts-4level.test
*** PASS: test_cases/q3/6-tiled-root.test
*** PASS: test_cases/q3/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q3/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent 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 AlphaBetaAgent on smallclassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases/q3/8-pacman-game.test

### Question q3: 5/5 ###

Finished at 22:13:32

Provisional grades
=====
Question q3: 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@051ab:~/Desktop/multiagents$
```

بخش چهارم

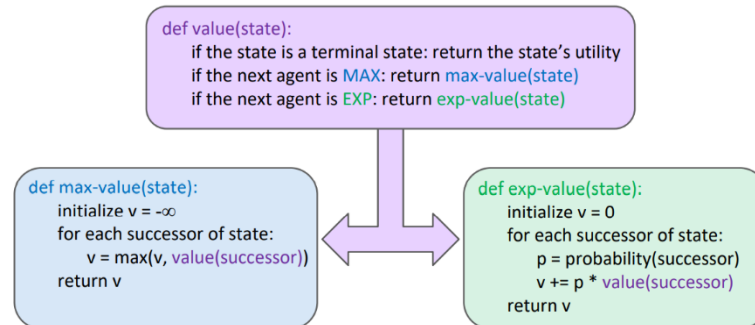
مینیماکس احتمالی

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

```
339 def getAction(self, gameState):
340     """
341     Returns the expectimax action using self.depth and self.evaluationFunction
342
343     All ghosts should be modeled as choosing uniformly at random from their
344     legal moves.
345     """
346     """ YOUR CODE HERE """
347     # we want to choose best action
348     # first we should find possible successors for this state of pacman
349     # then we should find best action to best successor for get maximum value
350     # for do this job we need find the value for possible successors with the help of value function(Expectimax algorithm)
351     # then we choose maximum value from them and return the action related to that
352     value_action = (-float("inf"), None)
353     for action in gameState.getLegalActions(self.index):
354         new_value_action = ((self.value(gameState.generateSuccessor(self.index, action), 1, 1)), action)
355         value_action = max(value_action, new_value_action, key=lambda x: x[0])
356     return value_action[1]
357
358 # according to slides our pseudocode for Expectimax function:
359 """
360 def value(state):
361     if the state is a terminal state: return the state's utility
362     if the next agent is MAX: return max value(state)
363     if the next agent is EXP: return exp_value(state)
364 """
365 def value(self, gameState, agentIndex, depth):
366     # first step: we check if the state is a terminal state or not
367     if gameState.iswin() or gameState.islose() or depth == self.depth * gameState.getNumAgents():
368         return self.evaluationFunction(gameState)
369     # second step: if the agentIndex == 0 so next agent is MAX
370     if agentIndex == 0:
371         return self.max_value(gameState, agentIndex, depth)
372     # third step: if the agentIndex > 0 so next agent is MIN
373     else:
374         return self.exp_value(gameState, agentIndex, depth)
375
376 # according to slides our pseudocode for max_value function:
377 """
378 def max_value(state):
379     initialize v = - inf
380     for each successor of state:
381         v = max(v, value(successor))
382 """
383 def max_value(self, gameState, agentIndex, depth):
384     # first step: initialize v = - inf
385     v = -float("inf")
386     # second step: find successors list
387     legal_actions = gameState.getLegalActions(agentIndex)
388     successor_list = []
389     for action in legal_actions:
390         successor = gameState.generateSuccessor(agentIndex, action)
391         successor_list.append(successor)
392     # third step: find maximum value and update v
393     for successor in successor_list:
394         v = max(v, (self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)))
395     return v
396
397 # according to slides our pseudocode for exp_value function:
398 """
399 def exp_value(state):
400     initialize v = 0
401     for each successor of state:
402         p = probability(successor)
403         v += p*value(successor)
404 """
405 def exp_value(self, gameState, agentIndex, depth):
406     # first step: initialize v = 0
407     v = 0
408     # second step: find successors list
409     legal_actions = gameState.getLegalActions(agentIndex)
410     successor_list = []
411     for action in legal_actions:
412         successor = gameState.generateSuccessor(agentIndex, action)
413         successor_list.append(successor)
414     # third step: find probability and update v
415     for successor in successor_list:
416         p = 1.0/len(legal_actions)
417         v += p * self.value(successor, (agentIndex+1)%gameState.getNumAgents(), depth+1)
418     return v
419
420
421
422
```

توضیح کد:

در اینجا نیز طبق شبهه 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 agent 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 از بین 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 هر کدام 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 value را در خروجی getAction برمیگردانیم.

امتحان عامل با دستورات گفته شده:

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

```
mahdi@CSLab:~/code$ 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@CSLab:~/Desktop/multiagents$
```



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



ارزیابی سوال چهارم:

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

```
mahdig05lab:~/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.

mahdig05lab:~/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
```

```
hahdi@OSLab:~/Desktop/multiagents$ python3 pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores: -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0
Win Rate: 0/10 (0.00)
Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
hahdi@OSLab:~/Desktop/multiagents$
```

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
```

```
hahdi@OSLab:~/Desktop/multiagents$ python3 pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Average Score: 221.8
Scores: 532.0, 532.0, 532.0, -502.0, -502.0, -502.0, 532.0, 532.0, 532.0, 532.0
Win Rate: 7/10 (0.70)
Record: Win, Win, Win, Loss, Loss, Loss, Win, Win, Win, Win
hahdi@OSLab:~/Desktop/multiagents$
```

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.

تابع ارزیابی

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

```

424 def betterEvaluationFunction(currentGameState):
425     """
426     Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
427     evaluation function (question 5).
428
429     DESCRIPTION: <write something here so we know what you did>
430     """
431     """ YOUR CODE HERE """
432
433     Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
434     evaluation function (question 5).
435     DESCRIPTION: <write something here so we know what you did>
436     In this evaluation function we want to consider : 1) distance to closest ghost
437     | 2) distance to closest food
438     | 3) number of remaining capsules
439     | 4) number of remaining food
440     | 5) game score
441
442     """
443     # like question1 we should first find useful information from GameState
444     pacman_pos = currentGameState.getPacmanPosition()
445     newFood = currentGameState.getFood()
446     ghost_positions = currentGameState.getGhostPositions()
447     # initialize our parameters:
448     # distance to ghost
449     closest_ghost_distance = 0
450     # distance to closest food
451     closest_food_distance = float('inf')
452     # number of remaining capsules
453     remain_capsule_num = 0
454     # number of remaining food
455     remain_food_num = 0
456     # game score
457     game_score = 0
458
459     # first we check if we close to a ghost we should return a very low value
460     ghost_distances = []
461     for ghost_pos in ghost_positions:
462         ghost_distance = manhattanDistance(pacman_pos, ghost_pos)
463         ghost_distances.append(ghost_distance)
464         if (ghost_distance == 0 or ghost_distance == 1):
465             return -float('inf')
466     if len(ghost_distances) != 0:
467         closest_ghost_distance = min(ghost_distances)
468
469     # second we want to calculate our distance to closest food
470     for food_pos in newFood.asList():
471         closest_food_distance = min(closest_food_distance, manhattanDistance(pacman_pos, food_pos))
472
473     # third we want to calculate number of remaining capsules
474     # we plus 1 because of avoiding to be 0 and getting divide by zero error
475     remain_capsule_num = len(currentGameState.getCapsules()) + 1
476
477     # fourth we want to calculate number of remaining food
478     # we plus 1 because of avoiding to be 0 and getting divide by zero error
479     remain_food_num = currentGameState.getNumFood() + 1
480
481     # fifth we want to calculate the score of game in this state
482     game_score = currentGameState.getScore()
483
484     # a vector that contains the parameters:
485     parameters = [closest_ghost_distance, 1.0 / closest_food_distance, 1.0 / remain_capsule_num, 1.0 / remain_food_num, game_score]
486
487     # a vector that contains the weights related to each parameter
488     weights = [1, 1000, 10000, 1000, 100]
489
490     # calculating evaluation value
491     value = 0
492     for parameter, weight in zip(parameters, weights):
493         value += parameter * weight
494
495     return value
496

```

توضیح کد:

این قسمت نیز تا حدودی شبیه سوال ۱ می باشد منتها برای اینکه دقیق تر و بهتر نتیجه بگیریم تعداد پارامترهای موثر در مقداری که تابع بر میگرداند را افزایش دادیم و همچنین برای آن ها وزن در نظر 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 کپسول های 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 منهن 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 منهن 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 برگردانیدیم. 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 و ۱۰۰۰۰ چون تقسیم بر تعداد 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 رسیدیم. ابتدا دلایلی مثل بالا در ذهن داشتم و کمی اعداد را بالا و پایین کردم تا به یک نتیجه متعادل و خوب رسیدیم.

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 حدودا 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

```
nahdi@OSLab:~/Desktop/multiagents$ python3 autograder.py -q q5 --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 12-13 at 0:01:07

Question q5
=====

Pacman emerges victorious! Score: 1175
Pacman emerges victorious! Score: 1173
Pacman emerges victorious! Score: 1162
Pacman emerges victorious! Score: 971
Pacman emerges victorious! Score: 1125
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 1361
Pacman emerges victorious! Score: 1173
Pacman emerges victorious! Score: 1368
Pacman emerges victorious! Score: 1296
Average Score: 1197.8
Scores: 1175.0, 1173.0, 1162.0, 971.0, 1125.0, 1174.0, 1361.0, 1173.0, 1368.0, 1296.0
Win Rate: 10/10 (1.00)
Record: Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q5/grade-agent.test (6 of 6 points)
*** 1197.8 average score (2 of 2 points)
*** Grading scheme:
*** < 500: 0 points
*** >= 500: 1 points
*** >= 1000: 2 points
*** 10 games not timed out (1 of 1 points)
*** Grading scheme:
*** < 0: fail
*** >= 0: 0 points
*** >= 10: 1 points
*** 10 wins (3 of 3 points)
*** Grading scheme:
*** < 1: fail
*** >= 1: 1 points
*** >= 5: 2 points
*** >= 10: 3 points

### Question q5: 6/6 ###

Finished at 0:01:14

Provisional grades
=====
Question q5: 6/6
-----
Total: 6/6

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

nahdi@OSLab:~/Desktop/multiagents$
```