

به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

مبانی و کاربردهای هوش مصنوعی (پاییز ۱۴۰۱)

گزارش پروژه - فاز دوم

استاد درس:

دکتر جوانمردی

نام دانشجو:

محمدجواد رضوانیان

مهلت ارسال تمرین: ۲ دی ماه ۱۴۰۱

سوال ۱

توضیحات مقدماتی انجام آزمایش

تابع عملکرد عامل ما، بر اساس ۷ پارامتر عمل می کند که هر کدام دارای ضربی از اعداد ثابت هستند که طی آزمایش کردن بدست آمده است (متدهای تعریف کننده پارامترهای زیر در ادامه خواهد آمد):

- Position: امتیاز منفی برای عدم حرکت عامل
- Score: امتیاز کسب شده در حرکت بعدی
- Food score: امتیاز فاصله نزدیک ترین نقطه
- Ghost score: امتیاز منفی نزدیک شدن روح
- Capsules score: امتیاز مثبت برای خوردن کپسول
- Scared score: امتیاز مثبت خوردن روح های ترسیده
- Scared time: امتیاز مثبت بابت زمان باقی مانده ترسیدن روح ها. (راستش این رو فقط بخاطر اینکه از همه عوامل استفاده کرده باشم نوشتم 😊) ولی بعدا دیدم که اگه یه ضربی بهش بدم توی عملکرد عامل تغییراتی ایجاد میشه مثل اولویت خوردن روح ها)

خروجی های آزمایش

بعد از انجام دادن ۱۷ آزمایش بر روی دستور زیر، این مقادیر بدست آمد:

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

با انجام آزمایشات مذکور، به ضرایب زیر رسیدیم:

$$Final\ score = position + (۱.۰۵\ Score) + (۱.۵\ food\ Score) + (۱.۲۸\ ghost\ score) + (۱.۸\ scared\ score) + (۰.۵\ capsule\ score) + (۱.۲\ scared\ time)$$

همانطور که مشاهده می کنید امتیاز خوردن روح ترسیده، خوردن نقاط و فاصله از روح ها بیشترین تأثیر را دارد. علت اصلی آن هم دوری کردن عامل از شکست خوردن است؛ عامل ترجیح میدهد که دیرتر به هدف برسد تا اینکه زودتر شکست بخورد. با انتخاب ضرایب بالا به خروجی زیر رسیدیم:

```
Question q1
=====
Pacman emerges victorious! Score: 1443
Pacman emerges victorious! Score: 1252
Pacman emerges victorious! Score: 1234
Pacman emerges victorious! Score: 1418
Pacman emerges victorious! Score: 1225
Pacman emerges victorious! Score: 1161
Pacman emerges victorious! Score: 1116
Pacman emerges victorious! Score: 1169
Pacman emerges victorious! Score: 1433
Pacman emerges victorious! Score: 1006
Average Score: 1245.7
Scores:      1443.0, 1252.0, 1234.0, 1418.0, 1225.0, 1161.0, 1116.0, 1169.0, 1433.0, 1006.0
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)
***      1245.7 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 ###
```

همانطور که مشاهده می‌کنید، عامل ما در تمام ۱۰ تست انجام شده، پیروز بوده و میانگین امتیاز کسب شده برابر با ۱۲۴۵.۷ است که مقدار مطلوبی برای عامل ما محسوب می‌شود. در آخرین تست عامل گوشه پایین زمین گیر افتاده و بدلیل اینکه عامل فاصله اش با آخرین نقطه باقی مانده از منظر فاصله منتهن تفاوتی ایجاد نمی‌کرد، در طی یک حرکت رفت و برگشت زمان زیادی را از دست داد. هنگامی که حریف (روح) ما مقداری فاصله منتهن آن تا عامل کاهش پیدا کرد، حرکت عامل برای فرار کردن تغییر کرد و به سمت نقطه مدنظر حرکت کرد و آخرین نقطه را هم بلعید و تست تمام شد.

جزئیات آزمایش:

ضرایب final score

```
finalScore = position + (1.05 * score) + (1.50 * foodScore) + (1.28 * ghostScore) + (1.8 * scaredScore) + (0.5 * capsuleScore) + (1.2 * scaredTimer)
```

تابع PositionEstimate

```
def positionEstimate(currentPos, newPos):
    if currentPos == newPos:
        return -100
    else:
        return 0
```

تابع ghostScoreEstimate

```
def ghostStateEstimate(newGhostStates, newPos, minDistance):
    activeGhostPositions = [i.getPosition() for i in newGhostStates if
i.scaredTimer == 0]
    ghostScore = 100
    maxGhostDistance = -1
    minGhostDistance = -1
    if newPos in activeGhostPositions:
        ghostScore = -1000
    activeGhostDistance = [manhattanDistance(newPos, i) for i in
activeGhostPositions]

    if activeGhostDistance:
        maxGhostDistance = max(activeGhostDistance)
        minGhostDistance = min(activeGhostDistance)
        if minGhostDistance <= 1:
            ghostScore = -1000
            minGhostDistance = -1
        else:
            if minGhostDistance < minDistance:
                ghostScore = -100
            ghostScore = maxGhostDistance
    return ghostScore
```

تابع foodScoreEstimate

```
def foodScoreEstimate(newFood, NewPositions):
    foodDistance = []

    for i in newFood.asList():
        if newFood[i[0]][i[1]]:
```

```

        foodDistance.append(manhattanDistance(NewPositions, i))

    foodScore = 0
    minDistance = 0
    if foodDistance:
        minDistance = min(foodDistance)
        if minDistance == 0:
            foodScore = 100
        else:
            foodScore = 10/minDistance
    return foodScore, minDistance

```

تابع scaredScoreEstimate

```

def scaredScoreEstimate(newGhostStates, minDistance, newPos):
    scaredGhostPositions = [i.getPosition() for i in newGhostStates if
                             i.scaredTimer != 0]
    scaredScore = 0
    if len(scaredGhostPositions) > 0:
        if newPos in scaredGhostPositions:
            scaredScore = 20000
        else:
            scaredGhostDistance = min([manhattanDistance(newPos, i) for i in
                                       scaredGhostPositions])
            if scaredGhostDistance < minDistance:
                scaredScore = minDistance - scaredGhostDistance
    return scaredScore

```

تابع capsuleScoreEstimate

```

def capsuleScoreEstimate(newCapsule, minDistance, newPos):
    capsuleScore = 0
    if len(newCapsule) >= 1:
        capsuleDistance = min([manhattanDistance(newPos, i) for i in
                               newCapsule])
        if newPos in newCapsule and capsuleDistance < minDistance:
            capsuleScore = 2000
    return capsuleScore

```

تابع scaredTiemEstimate

```

scaredTimer = sum(newScaredTimes)

```

بررسی تأثیر عواملی غیر یک راستا

در این پروژه، برای محاسبه تابع عملکرد، پارامترهای مثبت و منفی مختلفی وجود داشت که هر کدام به نوعی به روی یکدیگر تأثیر گذار بودند که در اینجا به بررسی آن‌ها می‌پردازیم.

عوامل امتیاز منفی

فاصله عامل ما از روح‌ها مهم‌ترین عاملی است که امتیاز تابع عملکرد را کاهش می‌دهد. برای محاسبه این مقدار، یک مقدار منفی به فاصله نزدیک‌ترین روح به عامل ما نسبت داده شده است؛ به طوری که اگر در حرکت بعدی عامل ما با روح برخورد می‌کند، امتیاز ۱۰۰۰- می‌گیرد. با در نظر گرفتن یک امتیاز منفی بسیار زیاد می‌توانیم تقریباً این پیش‌بینی را داشته باشیم که اگر راه نجاتی وجود داشته باشد عامل ما آن را در نظر می‌گیرد.

عوامل امتیاز مثبت

عامل ما دارای ۶ عامل مثبت است که توصیف آن در ابتدای گزارش آمد.

سوال ۲

خروجی آزمایش:

با در نظر گرفتن خروجی‌های ۴ و ۳، نتیجه زیر بدست آمد:

```
P2 ➤ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman died! Score: -492
Average Score: -492.0
Scores: -492.0
Win Rate: 0/1 (0.00)
Record: Loss

P2 ➤ python pacman.py -p MinimaxAgent -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
```

اما اگر عمق ۳ را چند بار تکرار کنیم به خروجی‌های مختلفی می‌رسیم. آزمایش شماره ۲:

```
P2 ➤ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores: 512.0
Win Rate: 1/1 (1.00)
Record: Win
```

آزمایش شماره ۳:

```
P2 ➤ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3
Pacman died! Score: -497
Average Score: -497.0
Scores: -497.0
Win Rate: 0/1 (0.00)
Record: Loss
```

پس متوجه می‌شویم که ممکن است که عامل ما در تمام حالت‌ها برنده نشود. (همانگونه که در دستور پروژه گفته شده بود)

حالا اگر عمق را برابر با ۱ قرار دهیم به خروجی زیر می‌رسیم (آزمایش ۶ بار تکرار شده است):

```

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores: 516.0
Win Rate: 1/1 (1.00)
Record: Win

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman emerges victorious! Score: 510
Average Score: 510.0
Scores: 510.0
Win Rate: 1/1 (1.00)
Record: Win

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores: 516.0
Win Rate: 1/1 (1.00)
Record: Win

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman emerges victorious! Score: 508
Average Score: 508.0
Scores: 508.0
Win Rate: 1/1 (1.00)
Record: Win

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman died! Score: -497
Average Score: -497.0
Scores: -497.0
Win Rate: 0/1 (0.00)
Record: Loss

P2 python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
Pacman emerges victorious! Score: 508
Average Score: 508.0
Scores: 508.0
Win Rate: 1/1 (1.00)
Record: Win

```

وضعیت خروجی پروژه در Autograde

هنگامی که پروژه را با تصحیح خودکار اجرا می‌کنیم، نتیجه تست‌ها به صورت زیر خواهد بود:

```
python autograder.py -q q2
```

```

Starting on 12-20 at 11:00:33

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

```

نتیجه آزمایش به صورت زیر گزارش شده است:

```

*** 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 15 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

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


Finished at 11:00:48

Provisional grades
=====
Question q2: 5/5
-----
Total: 5/5

```

عامل ما در این آزمایش شکست خورد اما بدلیل مطلوب بودن پیمایش‌های درخت، نتیجه مورد انتظار کسب شد.

عامل انتحاری!

وقتی که عامل ما می‌بیند که هیچ راه نجاتی ندارد و شکست او حتمی است، خود را به کام مرگ کشانده و با سرعت به سمت مرگ می‌شتابد. (یه نارنجک به خودش می‌بنده و ) اما علت اصلی این حرکت پک من این است که می‌بیند که به ازای هر state زندگی باید ۱- امتیاز کسب کند، ترجیح می‌دهد که با بالاترین امتیاز ممکن بازی را خاتمه دهد که در نتیجه باید سریع‌ترین راه مردن را انتخاب کند تا هزینه بیشتری بابت حیات خود ندهد.


```
P2 python 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
```

همانطور که در شکل بالا مشاهده می‌کنید نتیجه اینکه عامل ما تا عمق ۳ را به صورت تابع Minimax بررسی کند، این است که سریع‌ترین راه برای مرگ را انتخاب کند چون عامل ما فکر می‌کند که حریف او بدترین انتخاب را برای او رقم خواهد زد.

جزئیات توابع:

برای رسیدن به خروجی مدنظر، ما نیاز به سه تابع داریم؛ تابع minimax، تابع محاسبه حالت عامل (maxValue) و تابع محاسبه حالت روح‌ها (minValue). تابع minimax وظیفه اصلی اش پیدا کردن نوبت کسی است که باید حرکت کند تا ببیند که باید کدام یکی از توابع max یا min را صدا کند. اگر مقدار index برابر با ۰ باشد یعنی نوبت عامل ما است و در نتیجه باید تابع maxValue را صدا کند در غیر این صورت نوبت روح‌ها است که باید تابع minValue را صدا کند.

تابع minimax

```
def minimax(self, gameState, agentIndex, depth):
    if depth is self.depth * gameState.getNumAgents() \
        or gameState.isLose() or gameState.isWin():
        return self.evaluationFunction(gameState)
    if agentIndex is 0:
        return self.maxValue(gameState, agentIndex, depth)[1]
    else:
        return self.minValue(gameState, agentIndex, depth)[1]
```

تابع maxValue

```
def maxValue(self, gameState, agentIndex, depth):
    bestAction = ("max", -float("inf"))
    for action in gameState.getLegalActions(agentIndex):
        succAction =
        (action, self.minimax(gameState.generateSuccessor(agentIndex, action),
                               (depth + 1) % gameState.getNumAgents(),
                               depth+1))
        bestAction = max(bestAction, succAction, key = lambda x:x[1])
    return bestAction
```

تابع minValue

```
def minValue(self, gameState, agentIndex, depth):
    bestAction = ("min", float("inf"))
    for action in gameState.getLegalActions(agentIndex):
        succAction =
        (action, self.minimax(gameState.generateSuccessor(agentIndex, action),
                               (depth +
                               1)%gameState.getNumAgents(), depth+1))
        bestAction = min(bestAction, succAction, key=lambda x:x[1])
    return bestAction
```

سوال ۳

جزئیات پیاده‌سازی

کد کلاس محاسبه کننده‌ی آلفا بتا، به صورت زیر عمل می‌کند:

- تابع اصلی این کلاس، تابع `getAction()` است که خود دارای ۲ زیر تابع برای محاسبه مقادیر `max` و `min` است. تابع `getAction` به صورت پیش‌فرض مقدار مثبت و منفی بی‌نهایت را برای آلفا و بتا در نظر می‌گیرد (مقدار ۱ میلیون را برای بتا و منفی همان را برای آلفا). سپس با توجه به اقدام و شماره‌ی عامل تعیین می‌کند که الان کدام یک از دو زیر تابع مذکور باید اجرا شود.

خروجی اجرای کد به صورت زیر است:

```
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\3-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-tied-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
```

همانطور که در تصویر بالا مشاهده می‌کنید، تمام تست‌ها به خوبی انجام شده‌اند.

```
** 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 0 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 21:28:53

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.
```

نتیجه بدست آمده دارای میانگین امتیاز ۸۴.۰ بوده و نتیجه آن شکست عامل و کسب امتیاز ۵ از ۵ بوده است.

تابع maxValue:

```
def maxLevel(gameState, depth, alpha, beta):
    myDepth = depth + 1
    if gameState.isWin() or gameState.isLose() or myDepth == self.depth:
        return self.evaluationFunction(gameState)
    maximum = -1000000
    # 🍷 Pacman selection
    actions = gameState.getLegalActions(0)
    ins = alpha
    for action in actions:
        successor = gameState.generateSuccessor(0, action)
        maximum = max(maximum, minLevel(successor, myDepth, 1, ins, beta))
    # 🐱 Purring in this steps
    if maximum > beta:
        return maximum
    ins = max(ins, maximum)
    return maximum
```

همانطور که در بالا مشاهده می‌کنید، در ابتدا این تابع بررسی می‌کند که وضعیت بازی در حال اجرا است یا به پایان رسیده و یا به عمق دلخواه رسیدیم یا خیر. سپس بدلیل اینکه می‌خواهیم بعداً از مقدار آلفا و بتا استفاده کنیم، لازم است که از روی مقدار بیشینه (آلفا) یک نمونه داشته باشیم (ins = alpha) تا در طی تغییرات ایجاد شده، آلفای اصلی تغییری نکند. حالا چون قرار است که این تابع توسط عامل اصلی (پکمن) صدا زده شود، پس باید شماره عامل آن برابر با ۰ باشد.

در حلقه مذکور، چون که انتخاب‌ها به صورت بازگشتی انجام می‌شود، تابع minLevel() را هم صدا کرده و مقدار ۱ را به عنوان شماره عامل قرار می‌دهیم. حالا با استفاده از شرط بیشتر بودن مقدار بیشینه، میتوانیم درخت را هرس کنیم و مقدار بیشینه را از بین ins و maximum انتخاب کنیم.

تابع minValue:

```
def minLevel(gameState, depth, agentIndex, alpha, beta):
    minimum = 1000000
    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)
    actions = gameState.getLegalActions(agentIndex)
    ins = beta
    for action in actions:
        successor = gameState.generateSuccessor(agentIndex, action)
        # 🍷 Pacman selection
        if agentIndex == (gameState.getNumAgents() - 1):
            minimum = min(minimum, maxLevel(successor, depth, alpha, ins))
            if minimum < alpha:
                return minimum
            ins = min(ins, minimum)
        # 🐱 Ghosts Selection
        else:
            minimum = min(minimum,
minLevel(successor, depth, agentIndex + 1, alpha, ins))
            if minimum < alpha:
```

```

        return minimum
    ins = min(ins, minimum)
    return minimum

```

تابع `minLevel()` هم مشابه تابع `maxLevel()` عمل می‌کند با این تفاوت که به جای آلفا، بتا را قرار داده و به جای `maximum` مقدار `minimum` را قرار می‌دهیم.

پاسخ هرس درخت

بعد از اجرای هرس کردن، متوجه می‌شویم که هیچ کدام از شاخه‌ها هرس نمی‌شوند؛ چون هیچ جا شرط بزرگتر بودن آلفا از بتا محقق نمی‌شود. مقدار ریشه درخت‌ها برابر با ۵ بوده و عامل ما باید به سمت b_1 (سمت چپ) حرکت کند.

پاسخ تفاوت مقدار

الگوریتم‌های هرس، نمی‌توانند مقدار متفاوتی را در ریشه تولید کنند و صرفاً سرعت پردازش را بالا می‌برند (اگر قرار بود که مقادیر تغییر کند که دیگر به کار نمی‌آمد!). اما درباره گره‌های میانی ممکن است که مقادیر کاملاً صحیح نباشد؛ چون برخی از شاخه‌ها هرس می‌شوند، مقادیر برخی نودها هم با توجه به هرس انجام شده ممکن است که واقعی نباشند.

سوال ۴

جزئیات پیاده‌سازی

برای پیاده‌سازی این قسمت، ما به دو تابع کمکی نیاز داریم؛ تابع `max_value` و تابع `exp_value`. در ابتدا از تابع `max_value` استفاده می‌کنیم تا چک کنیم که آیا در وضعیت مناسب قرار داریم یا خیر (پیروزی، شکست و یا محدودیت عمق). سپس برای هر عملی یک وزن تعریف می‌کنیم به نام `weight` که بتوانیم مقدار آن را با مقادیر بدست آمده مقایسه کنیم. سپس تمام اقدامات ممکن را بررسی کرده و وزن همه آن‌ها را در آرایه‌ای شامل [وزن، عمل] به نام `sucsValue` ذخیره می‌کنیم. درایه صفر ام این آرایه وزن عمل و آرایه دوم عملی است که می‌توان با آن انجام داد را نشان می‌دهد. حالا چک می‌کنیم تا بیشترین وزن را پیدا کنیم و سپس درایه یک ام را به عنوان عمل برمی‌گردانیم.

این عمل را به صورت بازگشتی انجام داده تا هنگامی که به محدودیت عمق برسیم. احتمال هر کدام از اعمال هم با انجام تقسیم وزن عمل تقسیم بر تعداد اقدامات بدست می‌آید.

خروجی پیاده‌سازی بالا، به صورت زیر است:

```
P2: python autograder.py -q q4 --no-graphic
D:\Projects\Python\Pacman World\P2\autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib and slated for removal in Python 3.12; see the module's documentation for alternative uses
import imp
multiAgents.py:163: SyntaxWarning: "is" with a literal. Did you mean "=="?
if agentIndex is 0:
Starting on 12-23 at 22:52:22

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 0 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 22:52:23

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.
```

تابع `max_value`:

```
def max_value(gameState, depth):
    Actions = gameState.getLegalActions(0)
```

```

    if len(Actions) == 0 or gameState.isWin() or gameState.isLose() or depth == self.depth:
        return (self.evaluationFunction(gameState), None)
    # Initializing the game state 🤖
    weight = -(float("inf"))
    Act = None

    for action in Actions:
        sucsValue = exp_value(gameState.generateSuccessor(0, action), 1, depth)
        sucsValue = sucsValue[0]
        if(weight < sucsValue):
            weight, Act = sucsValue, action
    return (weight, Act)

```

تابع `exp_value`:

```

def exp_value(gameState, agentID, depth):
    Actions = gameState.getLegalActions(agentID)
    if len(Actions) == 0:
        return (self.evaluationFunction(gameState), None)

    l = 0
    Act = None
    for action in Actions:
        if(agentID == gameState.getNumAgents() - 1):
            sucsValue = max_value(gameState.generateSuccessor(agentID, action), depth + 1)
        else:
            sucsValue = exp_value(gameState.generateSuccessor(agentID, action), agentID + 1, depth)
        sucsValue = sucsValue[0]
        prob = sucsValue / len(Actions)
        l += prob
    return (l, Act)

```

دلیل تفاوت عملکرد الگوریتم minimax و minimax احتمالی:

برای فهم بیشتر این موضوع، یک سناریو دستور کار را تست می‌کنیم. خروجی به صورت زیر است:

```

P2: python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
D:\Projects\Python\Pacman World\P2\multiAgents.py:163: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if agentIndex is 0:
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
Win Rate:    0/10 (0.00)
Record:      Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

```

حالا به صورت احتمالی اجرا می‌کنیم:

```
P2 python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Average Score: -191.8
Scores: -502.0, 532.0, -502.0, -502.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0
Win Rate: 3/10 (0.30)
Record: Loss, Win, Loss, Loss, Win, Win, Loss, Loss, Loss, Loss
```

همانطور که مشاهده می‌کنید، عامل ما در ۳ بار اجرا پیروز شده است و احتمال پیروزی در این آزمایش برابر با ۳۰٪ است. با انجام این آزمایش متوجه می‌شویم که گزاره‌ی دستورکار درست نیست که لزوماً ۵۰٪ مواقع پیروز خواهیم بود. دلیل این اتفاق هم پاسخ به سوالی است که در قسمت دوم داده شد؛ عامل ما عملیات انتحاری را انتخاب می‌کند. اما با توجه به احتمال داده شده، عامل ما اگر حریف‌های کاملاً حرفه‌ای نداشته باشد (یعنی همیشه بدترین تصمیم را برای عامل ما نگیرند) ۳۰٪ احتمال دارد که پیروز شود.

سوال ۵

جزئیات پیاده‌سازی

خروجی اجرای کد ما به صورت زیر است:

```
P2: python autograder.py -q q5 --no-graphics
D:\Projects\Python\Pacman World\P2\autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of importlib and slated for removal in Python 3.12; see the module's documentation for alternative uses
  import imp
Starting on 12-23 at 23:16:48

Question q5
=====

Pacman emerges victorious! Score: 1311
Pacman emerges victorious! Score: 1299
Pacman emerges victorious! Score: 1189
Pacman emerges victorious! Score: 1031
Pacman emerges victorious! Score: 1243
Pacman emerges victorious! Score: 1148
Pacman emerges victorious! Score: 1142
Pacman emerges victorious! Score: 1327
Pacman emerges victorious! Score: 1160
Pacman emerges victorious! Score: 1116
Average Score: 1188.6
Scores:      1311.0, 1299.0, 1189.0, 1031.0, 1243.0, 1148.0, 1142.0, 1327.0, 1160.0, 1116.0
Win Rate:    10/10 (1.00)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win

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

Finished at 23:16:59

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

همانطور که مشاهده می‌کنید، میانگین امتیازهای کسب شده برابر با ۱۱۸۸.۶ است. اما نکته‌ای که در این سوال خیلی مهم است، تغییر انتخاب از اقدامات به حالات است. چون ملاک اصلی ما برای انتخاب حالت فعلی عامل است، برای همین باید توابع سوال اول به کلی تغییر می‌کرد.

یکی از مهم‌ترین ویژگی‌های این نوع پیاده‌سازی، استفاده از ویژگی حالت فعلی در مقابل اقدامات بعدی است. برای همین مقادیر مورد بررسی واقعی‌تر و دقیق‌تر است.

جزئیات پیاده‌سازی:

(نمیدونم چی بنویسم!)

```
""" YOUR CODE HERE """

foodList = newFood.asList()
foodDistance = [0]
for pos in foodList:
    foodDistance.append(manhattanDistance(newPos, pos))

""" Manhattan distance to each ghost from the current state """
ghostPos = []
for ghost in newGhostStates:
    ghostPos.append(ghost.getPosition())
ghostDistance = [0]
for pos in ghostPos:
    ghostDistance.append(manhattanDistance(newPos, pos))
```



```
numberOfPowerPellets = len(successorGameState.getCapsules())

score = 0
numberOfNoFoods = len(newFood.asList(False))
sumScaredTimes = sum(newScaredTimes)
sumGhostDistance = sum(ghostDistance)
reciprocalfoodDistance = 0
if sum(foodDistance) > 0:
    reciprocalfoodDistance = 1.0 / sum(foodDistance)

score += successorGameState.getScore() + reciprocalfoodDistance +
numberOfNoFoods

if sumScaredTimes > 0:
    score += sumScaredTimes + (-1 * numberOfPowerPellets) + (-1 *
sumGhostDistance)
else :
    score += sumGhostDistance + numberOfPowerPellets
return score
```