گزارش پروژه ۲

پک‌من چندعامله (Multiagents)

نام و نام خانوادگی: روژینا کاشفی

تاریخ: ۱۴۰۰/۹/۲۱

سوال اول

در این سوال از ما خواسته شده است که تابع evalutionfunction را با استفاده از action و حالت(State) ثانویه پیاده‌سازی کنیم.
میدانیم evalutionfunction مثل یک هیوریستیکی است که باعث میشود ما تا برگ درخت پیمایش نکنیم و با استفاده از تابع نوشته مقادیر را بدست آوریم.(راه‌حل برای حل مشکل resource limitation)
مطابق زیر پیاده سازی انجام شده است.

```python
def evaluationFunction(self, currentGameState, action):
    """
    Design a better evaluation function here.

    The evaluation function takes in the current and proposed successor
    GameStates (pacman.py) and returns a number, where higher numbers are better.

    The code below extracts some useful information from the state, like the
    remaining food (newFood) and Pacman position after moving (newPos).
    newScaredTimes holds the number of moves that each ghost will remain
    scared because of Pacman having eaten a power pellet.

    Print out these variables to see what you're getting, then combine them
    to create a masterful evaluation function.
    """
    # Useful information you can extract from a GameState (pacman.py)
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
    "*** YOUR CODE HERE ***"
    evf_result = successorGameState.getScore()
    food_distance = [manhattanDistance(newPos, i) for i in newFood.asList()]

    if action == "Stop":
        evf_result -= 150

    for i in range(len(newGhostStates)):
        if (newGhostStates[i].getPosition() == newPos and (newScaredTimes[i] == 0)) or \
                util.manhattanDistance(newGhostStates[i].getPosition(), newPos) < 2:
            evf_result -= 100
    if len(food_distance) > 0:
        evf_result += 1 / min(food_distance)

    return evf_result
```

مطابق روبه‌رو مشاهده میکنیم از action و successor ها(حالت‌های ثانویه) استفاده شده.

در این سوال یک مقدار evf_result بازگردانده میشود که ابتدا امتیاز successorها است و سپس بسته به اینکه اگر action ما stop باشه باید ازش کم شود تا احتمال انتخابش توسط پک من که max است کم شود.
همینطور اگر موقعیت پک من با موقعیت روحی برابر باشد و روح مورد نظر زمان ترسش صفر باشد و همچنین اگر روحی در فاصله نزدیک پک من بود باید از evf_resultکم شود تا پک من متوجه شود که در این دو حالت نرود.
اما اگر فاصله پک من با غذا کم باشد موجب میشود نتیجه تقسیم بزرگتر شود و برای پک‌من بهتر است پس بنابراین به evf_result اضافه میکنیم.

پس از پیاده سازی نقشه testclassic همواره برنده میشود که مشاهده میکنیم.

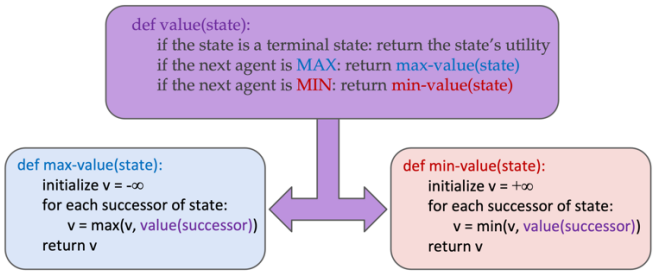
```
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p ReflexAgent -l testClassic
Pacman emerges victorious! Score: 564
Average Score: 564.0
Scores:        564.0
Win Rate:      1/1 (1.00)
Record:        Win
```

زمانی که تابع خود را برای دو حالت استفاده میکنیم برنده میشود.

```
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py --frameTime 0 -p ReflexAgent -k 1
Pacman emerges victorious! Score: 730
Average Score: 730.0
Scores:        730.0
Win Rate:      1/1 (1.00)
Record:        Win
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py --frameTime 0 -p ReflexAgent -k 2
Pacman emerges victorious! Score: 1586
Average Score: 1586.0
Scores:        1586.0
Win Rate:      1/1 (1.00)
Record:        Win
```

سوال دوم

در سوال ۲ از ما خواسته شده که تابع getAction در کلاس minimaxAgent پیاده سازی کنیم.
میدانیم در miniMax دنبال مقدار maxدر ریشه هستیم و از الگوریتم زیر استفاده میکنیم.




```python
def value(self, gameState, depth):
    if depth == self.depth * gameState.getNumAgents() \
            or gameState.isLose() or \
            gameState.isWin():
        return "", self.evaluationFunction(gameState)
    if depth % gameState.getNumAgents() != 0:
        return self.minvalue(gameState, depth)
    else:
        return self.maxvalue(gameState, depth)
```

```python
def minvalue(self, gameState, depth):
    legal_actions = gameState.getLegalActions(depth % gameState.getNumAgents())
    if len(legal_actions) == 0:
        return "", self.evaluationFunction(gameState)

    min_result = "", float("Inf")
    for action in legal_actions:
        successor = gameState.generateSuccessor(depth % gameState.getNumAgents(), action)
        result = self.value(successor, depth + 1)
        if result[1] < min_result[1]:
            min_result = (action, result[1])
    return min_result
```

```python
def maxvalue(self, gameState, depth):
    legal_actions = gameState.getLegalActions(0)
    if len(legal_actions) == 0:
        return "", self.evaluationFunction(gameState)
    max_result = "", float("-Inf")
    for action in legal_actions:
        successor = gameState.generateSuccessor(0, action)
        result = self.value(successor, depth + 1)
        if result[1] > max_result[1]:
            max_result = (action, result[1])
    return max_result
```

مطابق فوق تابع value را داریم که مقادیر را در terminalState ها (لایه برگ) یا اگر برنده و بازنده شدیم با استفاده ازمقدار evalution function را بر میگرداند.
میدانیم هر تعداد عامل داشته باشیم طول درخت ما ضرب در تعداد عامل ها میشود.مثلا اگر دو عامله باشد به ازای یک بار اجرای پک من یک بار اجرای عامل مقابل داریم که این موجب میشود طول درخت دو برابر شود.
اگر در terminalState نباشیم و یا در حالت برنده یا بازنده بودن در حال پیمایش تا رسیدن به برگ هستیم یا محاسبه Min و maxها در هر مرحله هستیم. میدانیم بعد از یک بار اجرا شدن عامل های مقابل نوبت به ما میرسد پس زمانی که depth % numOfAgents مساوی صفر شود یعنی دوباره نوبت پک‌من شده و ما به محاسبه ماکس میپردازیم در غیر این صورت نوبت عامل مقابل میشود که با تابع min کار داریم.

در تابع minValue ابتدا بسته به اینکه کدام عامل هست عمل‌هایی که میتوانیم انجام دهیم متفاوت است و اگر عملی نتوانیم انجام دهیم یعنی terminalState هستیم و مقدار evalutionFunction را برمیگرداند.
سپس بسته به عمل های مجازی که داریم مشاهده میکنیم فرزندان چی هستند و تا برگ به محاسبه value ادامه میدهیم(depth+1).
زمانی که به برگ رسیدیم مقایسه میکنیم نتیجه لایه برگ و پدری که دارد اگر کمتر بود جابه‌جا میکنیم. و به صورت بازگشتی بر میگردیم به سمت بالا. در انتها یک tuple از action و هزینه min برگردانده میشود.

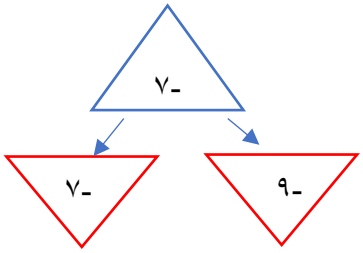
در مقابل max را داریم که چون میدانیم فقط pacman است اندیس صفر را برای پیداکردن عمل‌های مجاز میگذاریم.
و در بقیه جاها مطابق تابع min عمل میکند با این تفاوت بین پدرو فرزندان زمانی جابه‌جا میکند که مقدار فرزندان بیشتر از پدر باشند.

تابع getAction عملی که در مرحله باید پیمایش کند را بر میگرداند.

مطابق راهنمایی‌ها مشاهده میکنیم درعمق‌های ۱،۲،۳،۴ به ترتیب برابر۷،۸،۹ و -۴۹۲ است.

```
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=1
9.0
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=2
8.0
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=3
7.0
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
-492.0
```

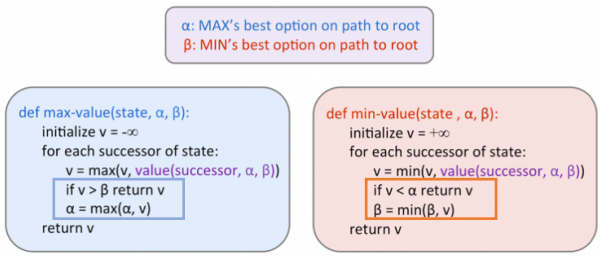
زمانی که پک‌من به این نتیجه برسد که مردنش اجتناب ناپذیر است، تلاش میکند که با امتیاز بهتر خود را بکشد و این رفتار خودکشی پک‌من است.
تصور کنید که پک من میان دو انتخاب زیر مانده است چون ماکس است مقدار بزرگتر را انتخاب میکند و اهمیتی ندارد امتیاز منفی است این کار موجب میشود با از دست دادن کمترین امتیاز بازی را انجام دهد.
با اجرای دستور زیر مشاهده میکنیم که خودکشی را بیشترین امتیاز انجام داد.




```
rojina@Rojinakashefis-MacBook-Pro multiagents % 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
```

سوال سوم

در سوال ۳ به دلیل محدودیت منابع که در روش minimax وجود داشت از روش آلفا بتا استفاده شده.در روش الفا بتا ما به هرس درخت minimax میپردازیم و جاهایی که ضروری نیست را بررسی نمی‌کنیم. مقادیر ماکسیمم در alphabeta مانند minimax خواهد بود اما مقدار مرزی متفاوت خواهد بود.




```
def getAction(self, gameState):
    """
    Returns the minimax action using self.depth and self.evaluationFunction
    """
    "*** YOUR CODE HERE ***"
    return self.value(gameState, 0, float("-Inf"), float("Inf"))[0]

def value(self, gameState, depth, alpha, beta):
    if depth == self.depth * gameState.getNumAgents() or gameState.isWin() or gameState.isLose():
        return "", self.evaluationFunction(gameState)
    if depth % gameState.getNumAgents() != 0:
        return self.minvalue(gameState, depth, alpha, beta)
    else:
        return self.maxvalue(gameState, depth, alpha, beta)
```

این قسمت مانند قسمت قبل است با این تفاوت که مقدار آلفا و بتا نیز پاس داده میشود که مقدار آلفا در ابتدا منفی بینهایت است تا هر مقداری بزرگتر وارد شد ان را بپذیرد و برای بتا نیز مثبت بی نهایت است که به این معناست هر مقداری کوچکتر از آن بود را بپذیرد.

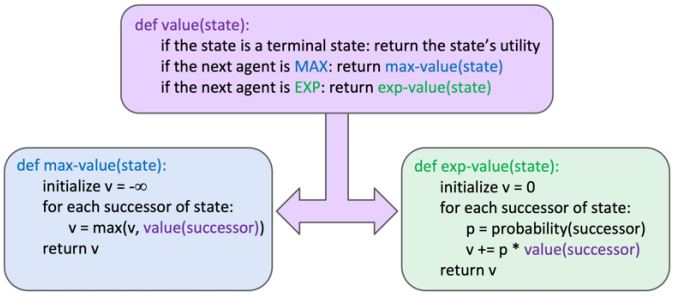
```python
def maxvalue(self, gameState, depth, alpha, beta):
    actions = gameState.getLegalActions(0)
    if len(actions) == 0:
        return "", self.evaluationFunction(gameState)
    max_result = ("", float("-Inf"))
    for action in actions:
        successor = gameState.generateSuccessor(0, action)
        result = self.value(successor, depth + 1, alpha, beta)
        if result[1] > max_result[1]:
            max_result = (action, result[1])
        if max_result[1] > beta:
            return max_result
        alpha = max(alpha, max_result[1])
    return max_result
```

```python
def minvalue(self, gameState, depth, alpha, beta):
    actions = gameState.getLegalActions(depth % gameState.getNumAgents())
    if len(actions) == 0:
        return "", self.evaluationFunction(gameState)
    min_result = ("", float("Inf"))
    for action in actions:
        successor = gameState.generateSuccessor(depth % gameState.getNumAgents(), action)
        result = self.value(successor, depth + 1, alpha, beta)
        if result[1] < min_result[1]:
            min_result = (action, result[1])
        if min_result[1] < alpha:
            return min_result
        beta = min(beta, min_result[1])
    return min_result
```

در این دو قسمت مقایسه با مقادیر آلفا و بتا اضافه شده است.

سوال چهارم

در روش های قبلی از adversarial search استفاده شده است که فرض برآن میشد عامل مقابل بهترین بازی خود را بازی میکند و هیچ گاه دچار اشتباه نمیشود.اما در واقعیت اینطور نیست و به دلایل مختلفی ممکن است بهترین بازی خود را نداشته باشد.

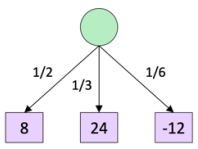


در روش minimax زمانی که در تله قرار میگرفت اقدام به خودکشی میکرد اما در این جا ۵۰ درصد احتمال میدهد که عامل مقابل اشتباه کند و پک‌من شانس برنده شدن دارد.

```python
def expectvalue(self, gameState, depth):
    legal_actions = gameState.getLegalActions(depth % gameState.getNumAgents())
    expect_value = 0
    if len(legal_actions) == 0:
        return "", self.evaluationFunction(gameState)
    probability = 1. / len(legal_actions)
    for action in legal_actions:
        successor = gameState.generateSuccessor(depth % gameState.getNumAgents(), action)
        result = self.value(successor, depth + 1)
        expect_value += result[1] * probability
    return "", expect_value
```

چون در صورت سوال گفته شده ارواح بین ۴ حرکت مجاز که به صورت یکنواخت و تصادفی هستند حرکت مورد نظر را انتخاب میکند.
پس احتمال ما تقسیم بر تعداد حرکت مجاز در هر مرحله میشود.

محاسبه احتمال وزن دار آن مانند مثال زیر



$$expect_{value} = \frac{1}{2} \times 8 + \frac{1}{3} \times 24 + \frac{1}{6} \times -12 = 10$$

مشاهده میکنیم که در alphabetaagent که هرس شده minimax است در تمامی بازی‌ها اقدام به خودکشی میکند اما در expectimax ۵۰ درصد مواقع میبرد.

```
rojina@Rojinakashefis-MacBook-Pro multiagents % python 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
Win Rate:     0/10 (0.00)
Record:       Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
rojina@Rojinakashefis-MacBook-Pro multiagents % python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: 15.0
Scores:       532.0, 532.0, -502.0, 532.0, -502.0, 532.0, -502.0, -502.0, -502.0, 532.0
Win Rate:     5/10 (0.50)
Record:       Win, Win, Loss, Win, Loss, Win, Loss, Loss, Loss, Win
rojina@Rojinakashefis-MacBook-Pro multiagents %
```

سوال پنجم

در قسمت پنجم باید یک evaluation function بنویسیم که وضعیت فعلی(current) استفاده میکند(برخلاف سوال ۱ از وضعیت ثانویه و action استفاده کرده است.
قسمت اول برای پیدا کردن موقعیت نسبت به غذا است.قسمت دوم موقعیت نسبت به روح و در اخر تعداد کپسول باقی مانده.

```python
pacman_position = currentGameState.getPacmanPosition()
ghost_position = currentGameState.getGhostPositions()[0]
scared_timer = currentGameState.getGhostStates()[0].scaredTimer
ghost_distance = manhattanDistance(ghost_position, pacman_position)
food_position = currentGameState.getFood().asList()

food_items = []
ghost_near = 0

for food in food_position:
    food_items.append(-1 * manhattanDistance(pacman_position, food))
if not food_items:
    food_items.append(0)

if ghost_distance == 0 and scared_timer == 0:
    ghost_near = -150
elif scared_timer > 0:
    ghost_near = -1 / ghost_distance

num_capsules = len(currentGameState.getCapsules())

return currentGameState.getScore() + ghost_near + num_capsules + max(food_items)
```