مبانی و کاربردهای هوش مصنوعی

پروژه‌ی دوم

تانیا جواهرپور ۹۷۳۳۰۱۷

پروژه مطابق دستورکار ارائه شده پیاده سازی شده است و برای تمامی بخش‌ها در کد کامنت و توضیحات واضح گذاشته شده است.

۱. عکس العمل:

برای این بخش همانطور که در دستور پروژه ذکر شده است باید با تکمیل تابع evaluationFunction یک تابع ارزیابی بنویسیم.

- **توضیح کد:**

عکس پایین قسمتی از این تابع است که از قبل نوشته شده بود.

```
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]
```

برای تکمیل این تابع چنین عمل کردیم:

ابتدا به بررسی روح‌ها میپردازیم، توجه داشته باشید در این بخش از پروژه ارزیابی بدون توجه به میزان ترسیدن روح‌ها انجام می‌گیرد؛ همچنین دلیل این که بررسی را با روح شروع کردیم این است که از آنجا که ما باید با استفاده از action عمل ثانویه را ارزیابی کنیم، پس اگر فاصله با روح کمتر از ۲ باشد علنا پکمن با شکست مواجه میشود و برای شکست منفی بی‌نهایت در نظر گرفته شده است.

سپس اگر اجرای تابع ادامه یافت یعنی این تضمین وجود دارد که پکمن ما توسط روح‌ها خورده نشده است و در نتیجه باید غذها را بخورد. اگر با انجام این عمل در state ثانویه تعداد غذاها کمتر شده باشد، یعنی با انجام این عمل غذایی خورده شده است؛ در نتیجه پکمن خوب است که این کار را انجام دهد که خروجی مثبت بی‌نهایت این مفهوم را می‌ساند.

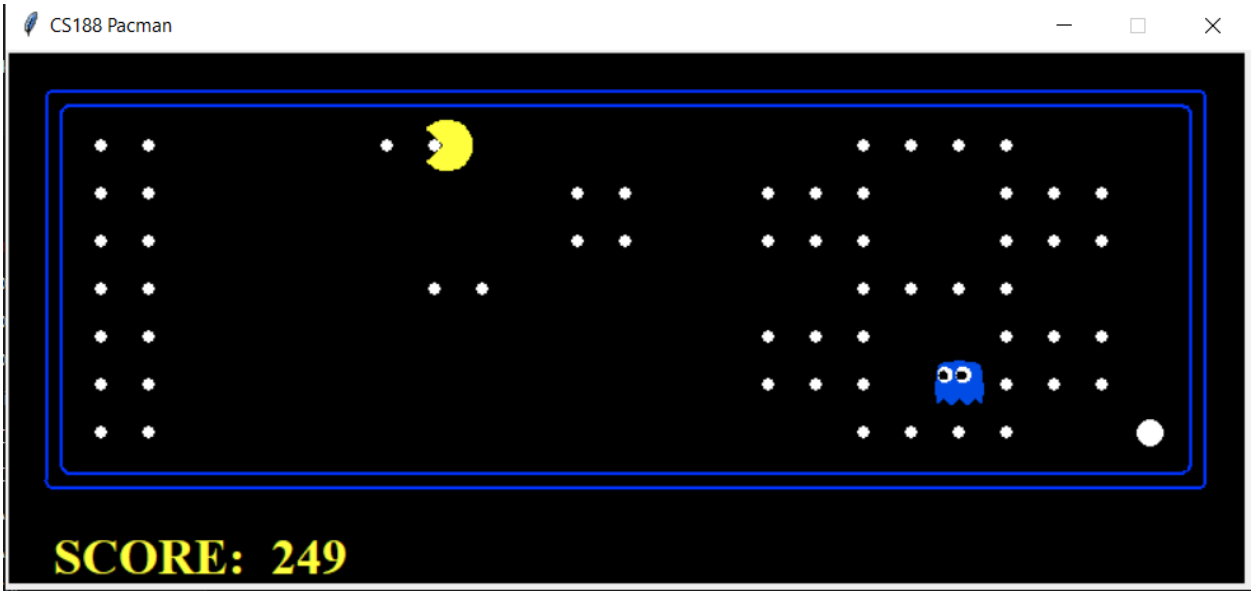
اگر تابع به اجرا ادامه دهد یعنی پکمن باید به دنبال غذایی که در فاصله‌ی دورتری قرار دارد بگردد، که منطقا بهترین غذا نزدیک‌ترین غذاست. به این منظور به دنبال نردیکترین غذا میگردیم و چون هر چه غذا نزدیکتر

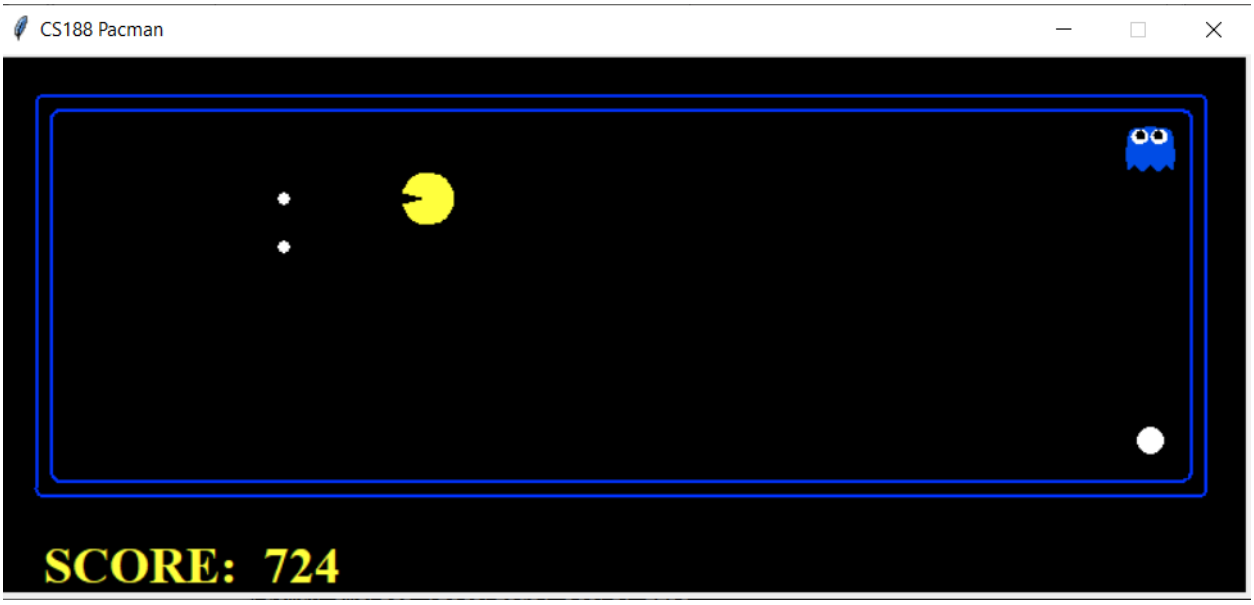
باشد بهتر و همر چه دورتر باشد بدتر است، خروجی برابر با عکس فاصله‌ی نزدیکترین غذا است.

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

# If agent takes this action there is a ghost that will eat the pacman,
# so pacman should run and that's why we give the lowest score.
for gp in newGhostStates:
    if manhattanDistance(newPos, gp.getPosition()) < 2:
        return -float('inf')

# By taking this action, no ghost will eat our pacman.
# If after doing this action the number of dots decreases,
# it implies by taking this action pacman will eat a dot.
# So do it and eat food!
currentFoodNum = len(currentGameState.getFood().asList())
newFoodList = newFood.asList()
newFoodNum = len(newFoodList)
if newFoodNum < currentFoodNum:
    return float('inf')

# Pacman could not eat food yet! So he should try to find the closest one.
# Find the closest food
minFoodDist = float('inf')
for food in newFoodList:
    foodDist = manhattanDistance(newPos, food)
    if foodDist < minFoodDist:
        minFoodDist = foodDist
# The lower distance is, the higher the score will be
return 1.0 / minFoodDist
```

برای این بخش میتوانستیم تابعی بسیار شبیه به تابع قسمت آخر پیاده کنیم، تنها با این تفاوت که بخش‌های newScaredTimes را حذف کنیم. (در این صورت ارزیابی دقیق‌تر بود ولی این تابع پیاده شده هم موارد خواسته شده و مورد ارزیابی را شامل می‌باشد.)

- تصاویری از خروجی این بخش:





- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\multiagents>python autograder.py -q q1
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 13:20:21

Question q1
===========

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1237
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1242
Average Score: 1240.2
Scores:        1238.0, 1244.0, 1239.0, 1240.0, 1239.0, 1237.0, 1238.0, 1247.0, 1238.0, 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)
***     1240.2 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 13:22:24

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

۲. مینیماکس

برای تکمیل این بخش باید در کلاس MiniMaxAgent به تکمیل تابع getAction میپرداختیم؛ به این منظور تابع miniMax در این کلاس پیاده شد. که عمق و عملگر را به عنوان ورودی می‌گیرد تا طبق تعریف تابع minimax و با توجه به این موضوع که پکمن باید مقدار بیشینه و روح‌ها باید مقدار کمینه را برگزینند، پیمایش صورت بگیرد.

- **توضیح کد:**

همانطور که گفته شد ابتدا تابع minimax صدا زده شده است و ورودی‌های مورد نیاز به آن داده شده‌است.

خروجی این تابع به صورت دوتایی عمل انتخاب شده و مقدار انتخابی می‌باشد که عمل انتخاب شده در اینجا و مقدار انتخابی برای مواردی که تابع به صورت بازگشتی صدا زده می‌شود تا پیمایش صورت بگیرد مورد نیاز است؛ این تابع به صورت مفصل توضیح داده خواهد شد.

همانطور که گفته شد چون خروجی این تابع به صورت دوتایی می‌باشد باید دومین خروجی که در واقع بهترین عمل انتخاب شده است را تابع getAction برگرداند.

```
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 ***"
    return self.minimax(gameState, self.depth, self.index)[1]
```

اگر پکمن برده باشد، یا باخته باشد و یا پیمایش را تمام کرده باشد و به عمق صفر رسیده باشد، باید تابع پایان یابد و مقدار انتخاب شده ارزیابی شود. در غیر این صورت نیاز است مقدار روح‌ها را داشته باشیم و با استفاده از این موضوع که عاملی که در حال بررسی آن هستیم پکمن است یا روح به اجرا ادامه دهیم.

همانطور که در خود کد هم گفته شده است عامل پکمن مقدار 0 و روح‌ها مقداری بیش از صفر دارند. هر بار که تابع اجرا می‌شود نیاز است تا عملگر بعدی مورد بررسی قرار بگیرد و با اضافه شدن 1 به مقدار متغیر agent نیاز است با محاسبه‌ی باقی‌مانده‌ی تقسیم آن بر مقدار کل عملگرها این که در حال بررسی کدام عملگر هستیم را استخراج کنیم.

```
    """
    Your minimax agent (question 2)
    """

    """
    Finds best action by iterating and using minimax algorithm
    """
    def minimax(self, gameState, depth, agent):

        # If Pacman wins or loses, game is finished ,
        # Also when maximum depth is reached means recursion should be stopped
        if depth == 0 or gameState.isLose() or gameState.isWin():
            return [self.evaluationFunction(gameState)]

        # One of the agents is pacman, so number of ghosts is all agents - pacman (1)
        ghostsNum = gameState.getNumAgents() - 1
        # Because agent increases each time
        agent = agent % (ghostsNum + 1)
```

ابتدا بخش مربوط به روح‌ها را توضیح می‌دهیم.

اگر agent مقداری بیش از 0 داشته باشد، عامل مورد بررسی روح است. اول این موضوع که این روح آخرین روح است چک میشود، تا در صورتی که بررسی تمام agentها در این عمق تمام شده بود، مقدار متغیر depth به اندازه‌ی یک واحد کاهش یابد. روی actionهای agent مورد نظر پیمایش انجام می‌دهیم تا مقدار کمینه را پیدا کنیم و آن را به صورت بهترین action برای این عملگر برگردانیم.

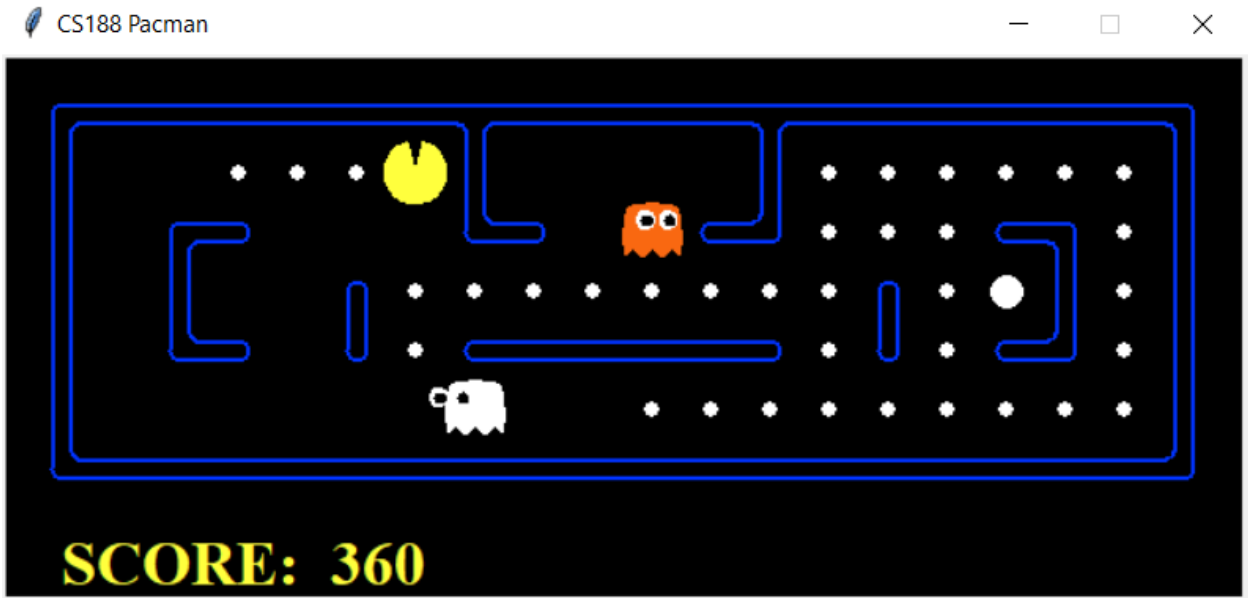
دوباره این موضوع قابل ذکر است که خروجی تابع به صورت دوتایی است که اولی آن مقدار انتخاب شده است.

```
# It's one of the ghosts' turn and min value should be selected.
if agent > 0:
    # If it's the final ghost, so depth should be decreased
    if agent == ghostsNum:
        depth -= 1

    minValue = float("inf")
    for action in gameState.getLegalActions(agent):
        successorGameState = gameState.generateSuccessor(agent, action)
        # Index 0 is the previous minValue and index 1 is the previous best Action
        # We should check next agent as next node
        newMinValue = self.minimax(successorGameState, depth, agent + 1)[0]

        # Update the minValue
        if newMinValue < minValue:
            bestAction = action
            minValue = newMinValue

    return minValue, bestAction
```

سپس در صورتی که تابع ادامه یابد یعنی مقدار agent برابر صفر بوده و عملگر ما پکمن است، پس باید مقدار بیشینه انتخاب شود. دوباره مانند بخش قبل روی عمل‌های آن پیمایش انجام میدهیم، با این تفاوت که این بار بهترین عمل آنی است که بیشترین مقدار خروجی را داشته باشد.

```
# It's pacman's turn and max value should be selected.
maxValue = -float("inf")
for action in gameState.getLegalActions(agent):
    successorGameState = gameState.generateSuccessor(agent, action)
    # We should check next agent as next node
    newMaxValue = self.minimax(successorGameState, depth, agent + 1)[0]

    # Update the maxValue
    if newMaxValue > maxValue:
        maxValue = newMaxValue
        bestAction = action

return maxValue, bestAction
```

- تصاویری از خروجی این بخش:



- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\multiagents>python autograder.py -q q2
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 17:13:44

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 34 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 17:14:18

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.
```

بررسی کنید چرا پکمن در این حالت به دنبال باخت سریع تر است.

چون وقتی مردن حتمی باشد، در واقع با دیرتر باختن فقط امتیاز پکمن کمتر خواهد شد، بنابر این باخت زودتر بهتر است.

```
C:\AI\AI_Projects\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
```

۳. هرس آلفا-بتا

برای تکمیل این بخش باید در کلاس AlfaBetaAgent به تکمیل تابع getAction میپرداختیم؛ به این منظور تابع alfaBeta در این کلاس پیاده شد.

- **توضیح کد:**

این تابع بسیار شبیه به تابع minimax است که در بخش قبل توضیح داده شد؛ تنها با این تفاوت که دو ورودی alfa و beta هم دارد و بر اساس تعریف هرس آلفا-بتا عمل میکند، به همین منظور این دو متغیر در ابتدا که تابع صدا زده میشود مقادیری برابر مثبت و منفی بی‌نهایت دارند.

```
def getAction(self, gameState):
    """
    Returns the minimax action using self.depth and self.evaluationFunction
    """
    "*** YOUR CODE HERE ***"
    alfa = -float("inf")
    beta = float("inf")
    return self.alphaBeta(alfa, beta, gameState, self.depth, self.index)[1]
```

بخش‌های ابتدایی دقیقا مشابه قسمت قبل است.

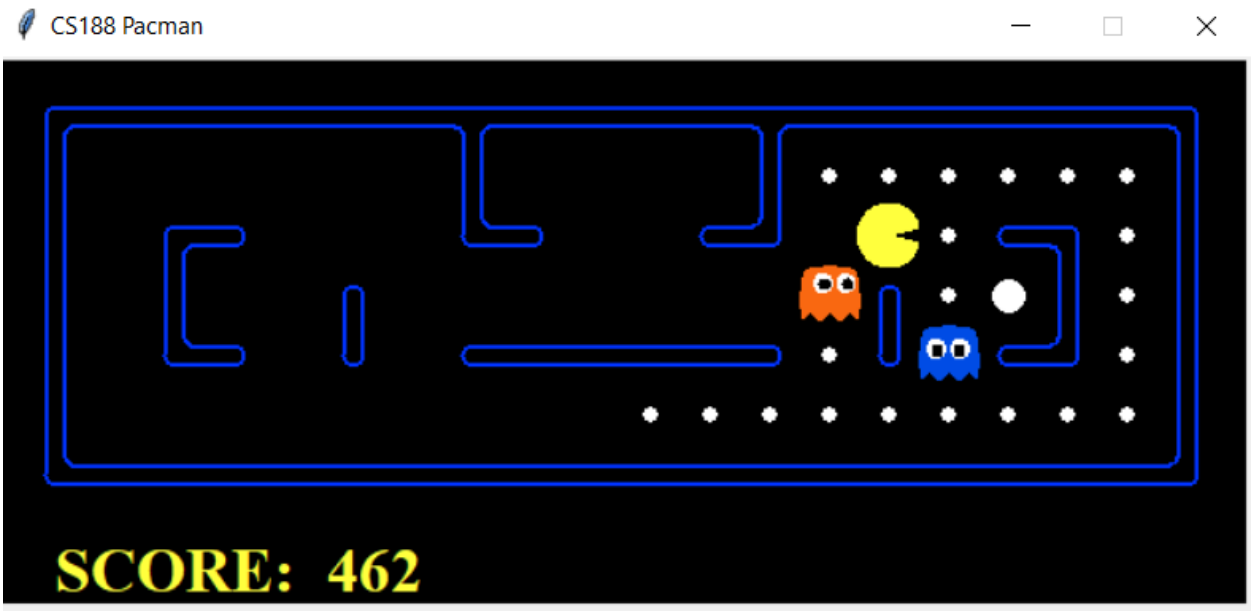
```
    """
    Finds best action by iterating and using alpha beta algorithm
    """
    def alphaBeta(self, alfa, beta, gameState, depth, agent):

        # If Pacman wins or loses, game is finished ,
        # Also when maximum depth is reached means recursion should be stopped
        if depth == 0 or gameState.isLose() or gameState.isWin():
            return [self.evaluationFunction(gameState)]

        # One of the agents is pacman, so number of ghosts is all agents - pacman (1)
        ghostsNum = gameState.getNumAgents() - 1
        # Because agent increases each time
        agent = agent % (ghostsNum + 1)

        # It's one of the ghosts' turn and min value should be selected.
        if agent > 0:
            # If it's the final ghost, so depth should be decreased
            if agent == ghostsNum:
                depth -= 1
```

همانطور که گفته شد خیلی شبیه به تابع minimax است تنها با این تفاوت که ابتدا هر بار که بسته به این که عملگر مورد بررسی پکمن است یا روح مقدار مینیمم یا ماکسیمم مشخص شده و سپس با آلفا و یا بتا مقایسه میشود. این بخش در کد زیر مشخص شده است.

```
# It's one of the ghosts' turn and min value should be selected.
if agent > 0:
    # If it's the final ghost, so depth should be decreased
    if agent == ghostsNum:
        depth -= 1

    minValue = float("inf")
    for action in gameState.getLegalActions(agent):
        successorGameState = gameState.generateSuccessor(agent, action)
        # Index 0 is the previous minValue and index 1 is the previous best Action
        # We should check next agent as next node
        newMinValue = self.alphaBeta(alfa, beta, successorGameState, depth, agent + 1)[0]

        # Update the minValue and beta in case needed
        if newMinValue < minValue:
            bestAction = action
            minValue = newMinValue

        if minValue < alfa:
            return minValue, bestAction

        beta = min(beta, minValue)

    return minValue, bestAction
```

```
# It's pacman's turn and max value should be selected.
maxValue = -float("inf")
for action in gameState.getLegalActions(agent):
    successorGameState = gameState.generateSuccessor(agent, action)
    # We should check next agent as next node
    newMaxValue = self.alphaBeta(alfa, beta, successorGameState, depth, agent + 1)[0]

    # Update the maxValue and alfa in case needed
    if newMaxValue > maxValue:
        bestAction = action
        maxValue = newMaxValue

    if maxValue > beta:
        return maxValue, bestAction

    alfa = max(alfa, maxValue)

return maxValue, bestAction
```

- تصاویری از خروجی این بخش:



- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\multiagents>python autograder.py -q q3
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 17:29:46

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-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
*** 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 31 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 17:30:18

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.


C:\AI\AI_Projects\multiagents>
```

۴. مینیماکس احتمالی

برای تکمیل این بخش باید در کلاس AlfaBetaAgent ExpectimaxAgent به تکمیل تابع getAction میپرداختیم؛ به این منظور تابع expectimax در این کلاس پیاده شد.

- **توضیح کد:**

این تابع بسیار شبیه به تابع minimax و alfaBeta در دو بخش قبلی است

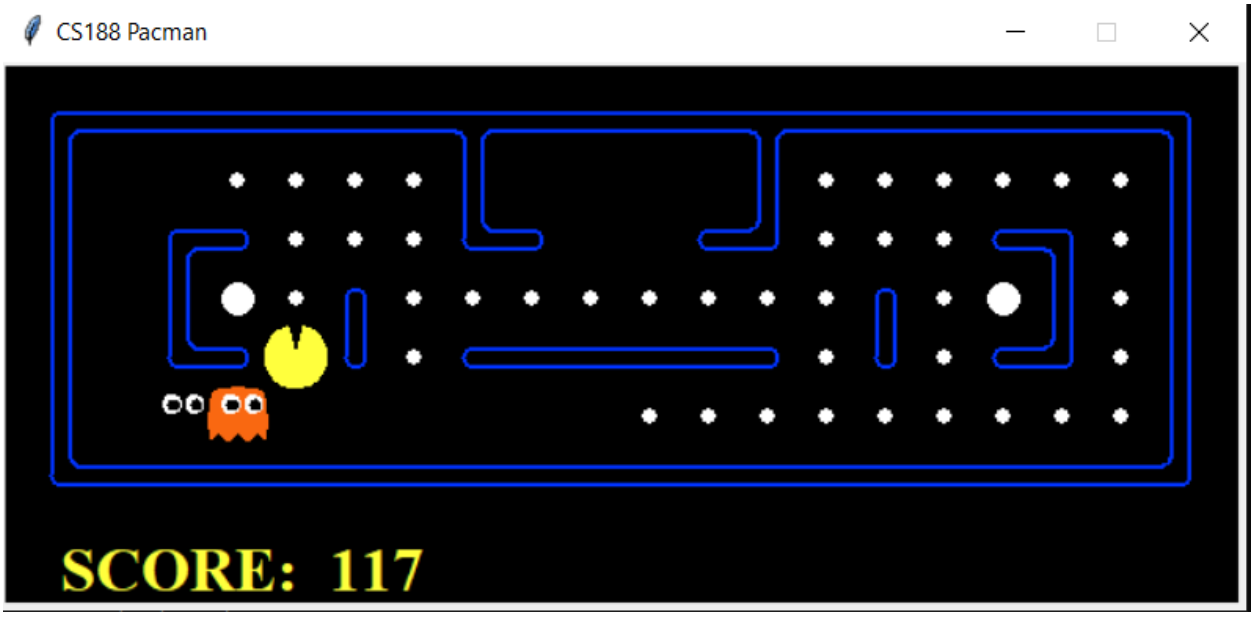
```
def getAction(self, gameState):
    """
    Returns the expectimax action using self.depth and self.evaluationFunction

    All ghosts should be modeled as choosing uniformly at random from their
    legal moves.
    """
    "*** YOUR CODE HERE ***"
    return self.expectimax(gameState, self.depth, self.index)[1]
```

```
  """
  Finds best action by iterating and using expectimax algorithm
  """
  def expectimax(self, gameState, depth, agent):

      # If Pacman wins or loses, game is finished ,
      # Also when maximum depth is reached means recursion should be stopped
      if depth == 0 or gameState.isLose() or gameState.isWin():
          return [self.evaluationFunction(gameState)]

      # One of the agents is pacman, so number of ghosts is all agents - pacman (1)
      ghostsNum = gameState.getNumAgents() - 1
      # Because agent increases each time
      agent = agent % (ghostsNum + 1)
```

برای این قسمت هم دقیقا طبق تعریف جلو رفتیم و برای پیدا کردن مقادیر مینیمم، در واقع مجموع مقادیر را در هر لایه از درخت تقسیم بر تعداد کردیم و action را به همراه این مقدار برگرداندیم.

```
    # It's one of the ghosts' turn and min value should be selected.
    if agent > 0:
        # If it's the final ghost, so depth should be decreased
        if agent == ghostsNum:
            depth -= 1

        minValues = 0
        for action in gameState.getLegalActions(agent):
            successorGameState = gameState.generateSuccessor(agent, action)
            # Index 0 is the previous minValue and index 1 is the previous best Action
            # We should check next agent as next node and sum all nodes' values
            # in order to take the average of all available utilities
            minValues += self.expectimax(successorGameState, depth, agent + 1)[0]

        # Calculate the average of all available utilities
        minValuesAvg = minValues / len(gameState.getLegalActions(agent))
        return minValuesAvg, action

    # It's pacman's turn and max value should be selected.
    maxValue = -float("inf")
    for action in gameState.getLegalActions(agent):
        successorGameState = gameState.generateSuccessor(agent, action)
        # We should check next agent as next node
        newMaxValue = self.expectimax(successorGameState, depth, agent + 1)[0]

        # Update the maxValue
        if newMaxValue > maxValue:
            bestAction = action
            maxValue = newMaxValue

    return maxValue, bestAction
```

- تصاویری از خروجی این بخش:



- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\multiagents>python autograder.py -q q4
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 17:40:08

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 32 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 17:40:41

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

۵. تابع ارزیابی:

این بخش خیلی شبیه به بخش اول پروژه است؛ ولی باید تابع ارزیابی نوشته شده دقیق‌تر باشد و states اهمیت پیدا می‌کند، به همین منظور ترسیدن روح‌ها هم اهمیت پیدا میکند.

- **توضیح کد:**

بخش اول تابع betterEvaluationFunction که در این بخش باید تکمیل گردد، بسیار شبیه به تابع evaluationFunction در بخش اول است که در آن غذاها، پوزیشن پکمن، موقعیت روح‌ها و تایم‌های ترسیدن آن‌ها در متغیرهای مربوطه ریخته می‌شوند.

```
def betterEvaluationFunction(currentGameState):
    """
      Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
      evaluation function (question 5).
      DESCRIPTION:
      The following features are considered and combined:
        - Compute the maze distance to the closest food dot
        - Compute the maze distance to the closest capsule
        - If the ghost is scared and close, eat it
        - If the ghost is not scared and close, run away
        - Take into account score (the longer the game is, the lower the score will be)
    """
    "*** YOUR CODE HERE ***"
    # Same evaluation function as evaluationFunction
    # If ghost is scared pacman runs to the ghost and eats the ghost
    newFood = currentGameState.getFood()
    newPos = currentGameState.getPacmanPosition()
    newGhostStates = currentGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
```

در بخش بعد ابتدا غذاها را به لیست تبدیل کردیم تا کار کردن با آن‌ها راحتتر گردد، سپس فاصله‌ی پکمن با نزدیکترین و دورترین غذا را به دست‌آوردیم.

همچنین از آنجا که روح‌ها هم در ارزیابی و حرکات پکمن تاثیرگذار هستند، فاصله‌ی نزدیکترین روح تا پکمن را نیز با پیمایش روی روح‌ها پیدا کردیم.

```
foodList = newFood.asList()

# There is no food, so the game is finished and the score will not chang
if not foodList:
    return currentGameState.getScore()

# Find the closest and the furthest food
minFoodDist = float('inf')
maxFoodDist = -float('inf')
for food in foodList:
    foodDist = manhattanDistance(newPos, food)
    if foodDist < minFoodDist:
        minFoodDist = foodDist
    if foodDist > maxFoodDist:
        maxFoodDist = foodDist

# Find the closest ghost from the Pacman
minGhostDist = float('inf')
for gp in newGhostStates:
    ghostDist = manhattanDistance(newPos, gp.getPosition())
    if ghostDist < minGhostDist:
        minGhostDist = ghostDist
```

اکنون نوبت محاسبه‌ی امتیاز و ارزیابی است.

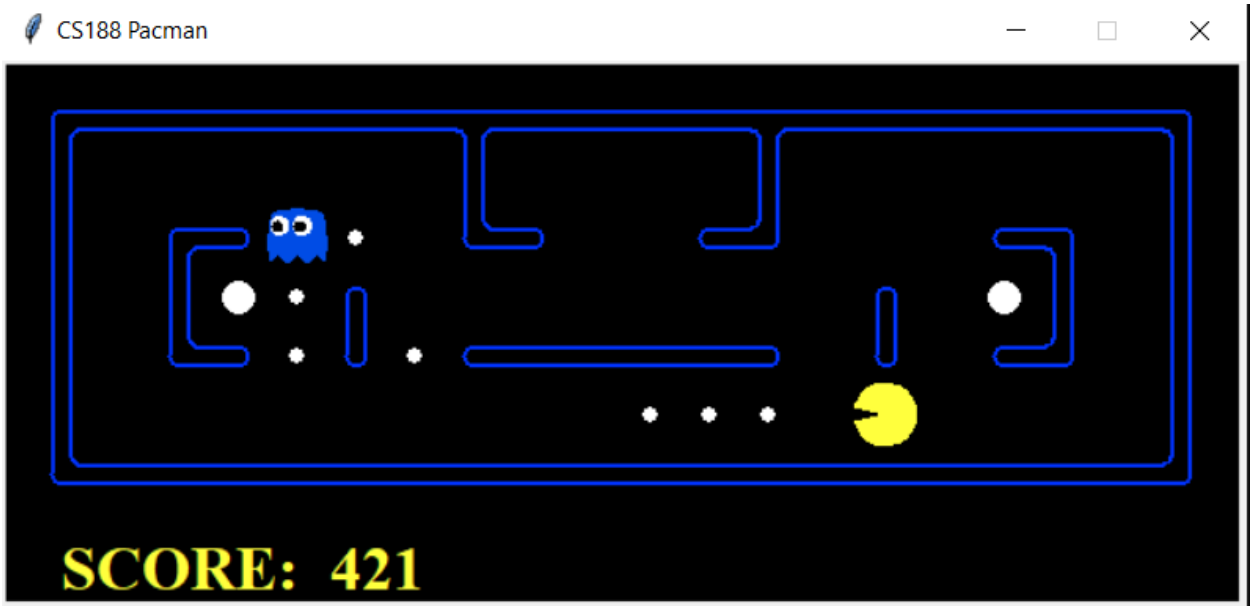
ابتدا ترسیدن روح‌ها را بررسی می‌کنیم، اگر روحی ترسیده باشد پکمن آن را میخورد؛ پس امتیاز به اندازه‌ی مجموع نزدیکترین غذا و نزدیکترین روح، کاهش می‌یابد.

سپس به بررسی غذاهای باقی‌مانده میپردازیم. اگر تنها یک غذا باقی‌مانده باشد، در واقع آن نقطه هم نزدیکترین غذا است و هم دورترین. در اینجا فاصله‌ی نزدیکترین روح اثرگذار است؛ اگر فاصله‌ی روح از پکمن، بیشتر از فاصله‌ی نزدیکترین غذا تا پکمن باشد ابتدا غذا خورده میشود، پس باید امتیاز افزایش یابد. اما اگر فاصله‌ی غذا تا او بیشتر از فاصله‌ی نزدیکترین روح باشد، این مفهموم دریافت میشود که برای رسیدن به نزدیکترین غذا، که درواقع تنها غذای باقی‌مانده است، خطر روح پکمن را تهدید میکند، به همین منظور امتیاز باید به میزان فاصله‌ی بین آن‌ها کاهش یابد.

در بخش آخر حالتی که بیش از یک غذا باقی‌مانده است بررسی شده است. تحلیل این بخش خیلی مانند بخش قبلی است با این تفاوت که چون بیش از یک غذا در زمین است، دو فاکتور مهم برای ارزیابی بر پایه‌ی غذاها داریم که یکی نزدیکترین آن‌ها و دیگری دورترینشان است. مجموع فاصله‌ی این دو از پکمن اهمیت دارد و این بار اختلاف مجموع فاصله‌ی آن‌ها با نزدیکترین روح است که برای ارزیابی

```
# Calculate new score based on states
score = currentGameState.getScore()

# If Ghost is Scared, pacman runs to the ghost to eat that ghost
if newScaredTimes[0] > 0:
    score += - minGhostDist - minFoodDist

# If there is only one dot left then minFoodDist == maxFoodDist
# If remained food is before the nearest ghost, the score increases,
# otherwise pacman should pass the post first and obviously the score decreases
# The lower food distance is, the higher the score will be
elif len(foodList) == 1:
    score += minGhostDist - minFoodDist

# There is more than one dot the score will change based on
# nearest and furthest distance between pacman and remained dots
else:
    score += minGhostDist - (maxFoodDist + minFoodDist)

return score
```

- تصاویری از خروجی این بخش:



- نتیجه‌ی خروجی autograder:

```
C:\AI\AI_Projects\multiagents>python autograder.py -q q5
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 18:00:00

Question q5
===========

Pacman emerges victorious! Score: 986
Pacman emerges victorious! Score: 977
Pacman emerges victorious! Score: 1346
Pacman emerges victorious! Score: 1181
Pacman emerges victorious! Score: 942
Pacman emerges victorious! Score: 1167
Pacman emerges victorious! Score: 1104
Pacman emerges victorious! Score: 1276
Pacman emerges victorious! Score: 1058
Pacman emerges victorious! Score: 947
Average Score: 1098.4
Scores:        986.0, 977.0, 1346.0, 1181.0, 942.0, 1167.0, 1104.0, 1276.0, 1058.0, 947.0
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)
***     1098.4 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 18:03:20

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

نتیجه‌ی نهایی autograder برای کل پروژه:

```
Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 5/5
Question q5: 6/6
------------------
Total: 25/25
```