به نام خدا

درس هوش مصنوعی و سیستم های خبره

تمرین پنجم

مدرس : دکتر محمدی

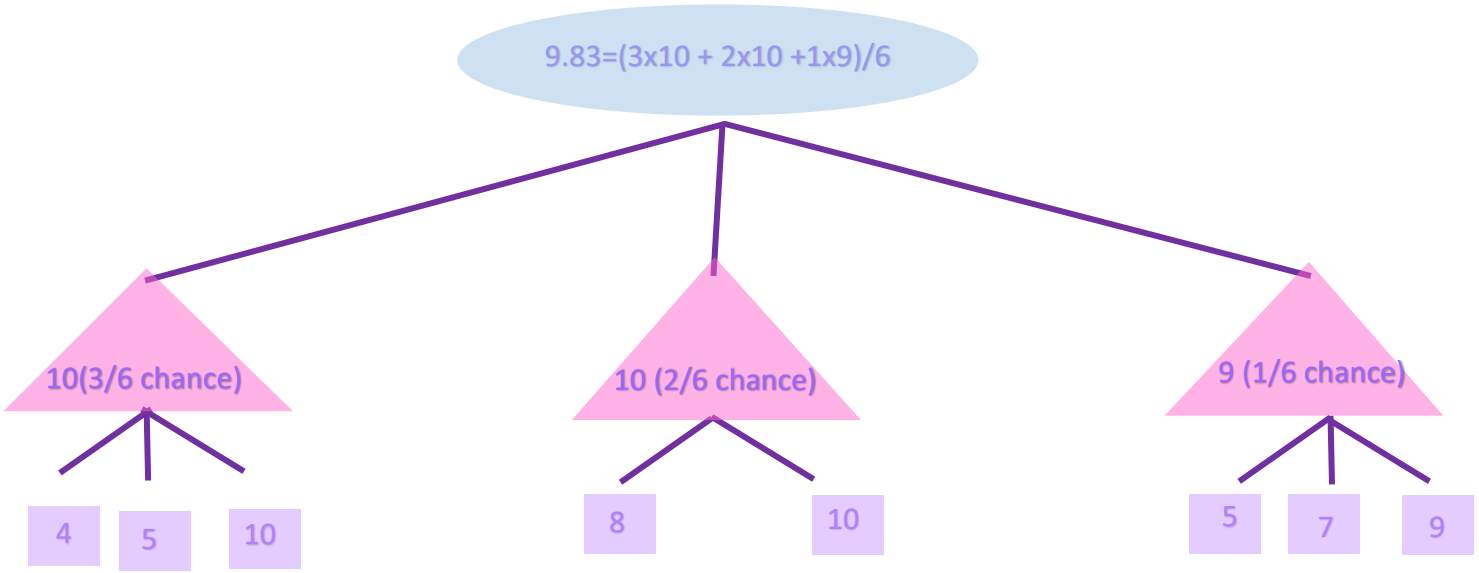
دانشجو : سارا سادات یونسی / ۹۸۵۳۳۰۵۳

۱. فرض کنید در یک بازی، هر بازیکن برای انجام یک حرکت باید یک تاس چند بعدی بیاندازد .درصورتی که نتیجه پرتاب تاس عددی فرد باشد، بازیکن سه حرکت احتمالی دارد که او را به حالت های سودمندی ۵، ۴و ۱۰می رساند. درصورتی که نتیجه پرتاب تاس ۲ یا ۴باشد، بازیکن دو حرکت احتمالی دارد که او را به حالت های سودمندی ۸و ۱۰می رساند. در نهایت اگر نتیجه عدد ۶باشد، سه حرکت وجود دارد که او را به حالت های سودمندی ۷، ۵و ۹می رساند .با توجه به توضیحات بازی فوق، درخت ExpecTiminiMax مربوطه را رسم کرده و مقدار گره ریشه آن را در هریک از دو حالت زیر تعیین کنید: (۲۰ نمره)

آ) بازیکن حداکثر(MAX) باشد.

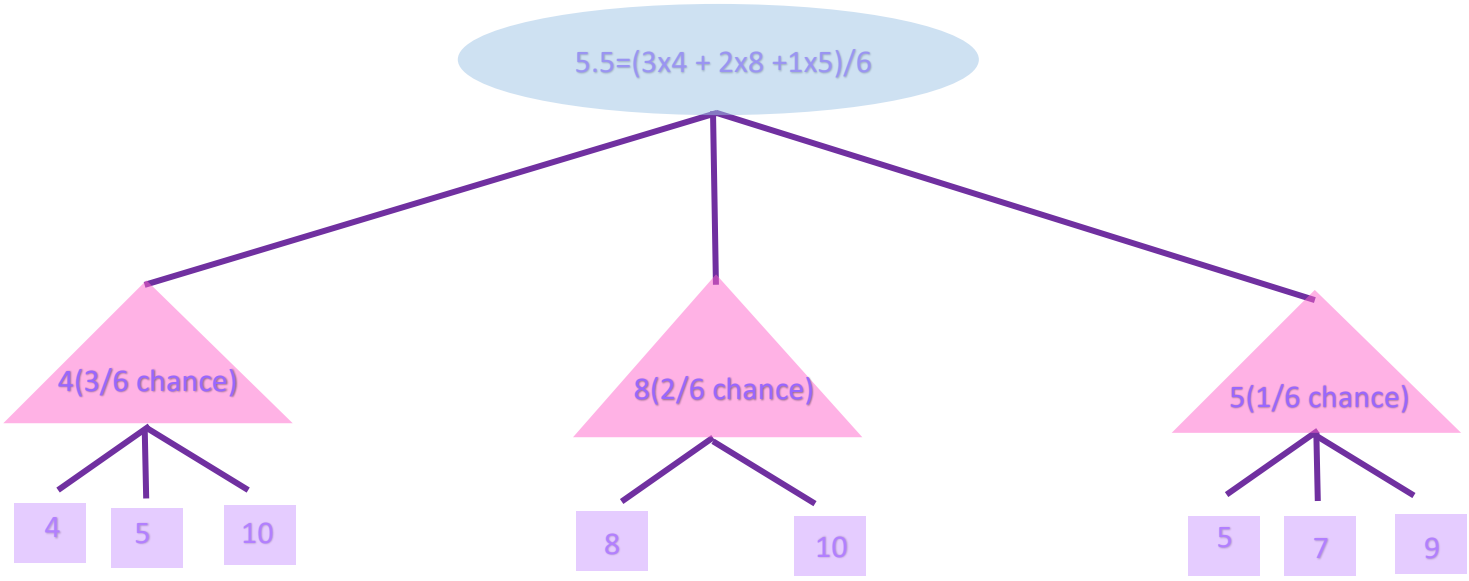
ب) بازیکن حداقل(MIN) باشد.

جواب الف)



برای گره های maximizer باید leaf ای انتخاب شود که مقدار ان بین فرزندان آن نود بیشینه باشد سپس این مقادیر را در مثلث های صورتی می نویسم سپس با توجه به هر یک از فرض های سوال و بررسی حرکت های احتمالی برای هر یک از حالت های مثلثمان یک احتمال (chance) در نظر گرفتیم که با ضرب حالت های احتمالی در وزن هر یک از ان ها و در نهایت مقدار گره ریشه را پیدا می کنیم . در نتیجه با پیدا کردن مقدار امید ریاضی و وزن های آن به جواب نهایی برسیم.

جواب ب)



برای گره های minimizer باید leaf ای انتخاب شود که مقدار ان بین فرزندان آن نود کمینه باشد سپس این مقادیر را در مثلث های صورتی می نویسم سپس با توجه به هر یک از فرض های سوال و بررسی حرکت های احتمالی برای هر یک از حالت های مثلثمان یک احتمال (chance) در نظر گرفتیم که با ضرب حالت های احتمالی در وزن هر یک از ان ها و در نهایت مقدار گره ریشه را پیدا می کنیم . در نتیجه با پیدا کردن مقدار امید ریاضی و وزن های آن به جواب نهایی برسیم.

بخش تحلیل نتایج کد ها :

- Reflex Agent

```
class ReflexAgent(Agent):
    """
    A reflex agent chooses an action at each choice point by examining
    its alternatives via a state evaluation function.

    The code below is provided as a guide.  You are welcome to change
    it in any way you see fit, so long as you don't touch our method
    headers.
    """

    def getAction(self, gameState):
        """
        You do not need to change this method, but you're welcome to.

        getAction chooses among the best options according to the
evaluation function.

        Just like in the previous project, getAction takes a GameState
and returns
        some Directions.X for some X in the set {NORTH, SOUTH, WEST,
EAST, STOP}
        """
        # Collect legal moves and successor states
        legalMoves = gameState.getLegalActions()

        # Choose one of the best actions
        scores = [self.evaluationFunction(
            gameState, action) for action in legalMoves]
        bestScore = max(scores)
        bestIndices = [index for index in range(
            len(scores)) if scores[index] == bestScore]
        # Pick randomly among the best
        chosenIndex = random.choice(bestIndices)

        "Add more of your code here if you want to"

        return legalMoves[chosenIndex]

    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]
        res = []
        for food in newFood.asList():
            res.append(1 / util.manhattanDistance(food, newPos))

        "*** YOUR CODE HERE ***"
        score = successorGameState.getScore()
        return score if len(res) == 0 else score + max(res)

def scoreEvaluationFunction(currentGameState):
    """
    This default evaluation function just returns the score of the state.
    The score is the same one displayed in the Pacman GUI.

    This evaluation function is meant for use with adversarial search agents
    (not reflex agents).
    """
    return currentGameState.getScore()

class MultiAgentSearchAgent(Agent):
    """
    This class provides some common elements to all of your
    multi-agent searchers.  Any methods defined here will be available
```

```
    to the MinimaxPacmanAgent, AlphaBetaPacmanAgent &
ExpectimaxPacmanAgent.

    You *do not* need to make any changes here, but you can if you want
to
    add functionality to all your adversarial search agents.  Please do
not
    remove anything, however.

    Note: this is an abstract class: one that should not be
instantiated.  It's
    only partially specified, and designed to be extended.  Agent
(game.py)
    is another abstract class.
    """

    def __init__(self, evalFn='scoreEvaluationFunction', depth='2'):
        self.index = 0  # Pacman is always agent index 0
        self.evaluationFunction = util.lookup(evalFn, globals())
        self.depth = int(depth)
```

تحلیل سوال یک :

- در این سوال به پیاده سازی Reflex Agent می پردازیم

در این سوال نزدیک ترین فاصله foods را پیدا می کنیم و سپس بررسی می کنیم اگر goast بین ما تا هدف وجود داشت در ان صورت فاصله تا هدف را بیش تر کند .

با استفاده از ManhatanDistance فاصله هر یک از روح ها نسبت به pacman بدست می آوریم و اگر صفر باشد می دانیم که به دشمن برخورد کرده ایم.

همچنین باید action چک شود که stop است یا نه چون در صورت stop کردن پک من حرکتی نخواهد کرد .

با ترکیب دو مقدار فاصله تا نزدیک ترین foods و فاصله تا نزدیک ترین goast، مقدار ارزش state که در حال حاضر در آن را بر میگردانیم.

- Minimax

```
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (question 2)
    """
    action = None
    all_agent_num = 0

```

```
    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 ***"
        self.action = None
        self.all_agent_num = gameState.getNumAgents()
        minimax_value = self.MiniMaxFunc(gameState, 0, 0)
        return self.action
        util.raiseNotDefined()

    def MiniMaxFunc(self, gameState, turn, depth):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        agent_id = turn % self.all_agent_num
        if agent_id == 0:
            if depth >= self.depth:
                return self.evaluationFunction(gameState)
            else:
                return self.max_value(gameState, turn, depth)
        else:
            return self.min_value(gameState, turn, depth)

    def max_value(self, gameState, turn, depth):
        v = -9999999
        legal_action = gameState.getLegalActions(0)
        for action in legal_action:
```

```
            state = gameState.generateSuccessor(0, action)
            maximum = self.MiniMaxFunc(state, turn + 1, depth + 1)
            if maximum > v:
                v = maximum
                if turn == 0:
                    self.action = action
        return v

    def min_value(self, gameState, turn, depth):
        v = +9999999
        index = turn % self.all_agent_num
        legal_action = gameState.getLegalActions(index)
        for action in legal_action:
            state = gameState.generateSuccessor(index, action)
            v = min(v, self.MiniMaxFunc(state, turn + 1, depth))
        return v
```

تحلیل سوال دو :

- در این سوال به پیاده سازی minimax می پردازیم

در این سوال به بررسی الگوریتم minimax می پردازیم :

```
function minimize(state):

  if is_terminal(state):
    return (NULL, eval(state))
  (min_state, min_score) = (NULL, ∞)

  for child in state.children():
    (_, score) = maximize(child)
    if score < min_score:
      (min_state, min_score) = (child, score)

  return (min_state, min_score)
```

در اینجا یا نوبت بازیکن max است حالتی که می خواهیم به ماکسیمم برسد.

یا بازیکن مین min است حالتی که می خواهیم به مینیموم برسد(دشمن).

یا به حالت پایانی میرسیم.

pacman باشد، با صدا زدن تابع maximizer ،مقدار بیشینه را از میان فرزندان آن بدست میآوریم و آن را باز می گردانیم.

در غیر این صورت نیاز است که با صدا زدن تابع minimizer ،مقدار کمینه را از میان فرزندانش بد ست آوریم و باز گردانیم.

Alpha-Beta pruning •

```
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (question 3)
    """
    action = None
    all_agent_num = 0

    def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and
self.evaluationFunction
        """
        self.all_agent_num = gameState.getNumAgents()
        self.alpha_beta(gameState, 0, 0, -999999, +999999)
        return self.action
        util.raiseNotDefined()

    def alpha_beta(self, gameState, turn, depth, alpha, beta):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        agent_id = turn % self.all_agent_num
        if agent_id == 0:
            if depth >= self.depth:
                return self.evaluationFunction(gameState)
            else:
                return self.max_value(gameState, turn, depth, alpha, beta)
        else:
            return self.min_value(gameState, turn, depth, alpha, beta)

    def max_value(self, gameState, turn, depth, alpha, beta):
        v = -9999999
        legal_action = gameState.getLegalActions(0)
        for action in legal_action:
            state = gameState.generateSuccessor(0, action)
            maximum = self.alpha_beta(state, turn + 1, depth + 1, alpha,
beta)
            if maximum > v:
                v = maximum
                if turn == 0:
                    self.action = action
            if v > beta:
                return v
```

```
            alpha = max(v, alpha)

        return v

    def min_value(self, gameState, turn, depth, alpha, beta):
        v = +9999999
        index = turn % self.all_agent_num
        legal_action = gameState.getLegalActions(index)
        for action in legal_action:
            state = gameState.generateSuccessor(index, action)
            v = min(v, self.alpha_beta(state, turn + 1, depth, alpha, beta))
            if v < alpha:
                return v
            beta = min(v, beta)
        return v
```

تحلیل سوال سه :

- در این سوال به پیاده سازی Alpha-Beta pruning می پردازیم

به بررسی الگوریتم Alpha-Beta pruning می پردازیم

این بخش هم همانند پیاده سازی minimax هست با این تفاوت که در بدنه ی minimizer و maximizer نیاز به بررسی و بروزرسانی مقادیر آلفا و بتا داریم.

در بخش minimax دو مقدار الفا و بتا به func به عنوان ارگومان های ورودی می دهیم .که maxو min مقدار از روت تا گره حال حاظر را نشان می دهد.

همواره پس از اپدیت چک می کنیم که به استیت پایانی رسیدیم یا نه.

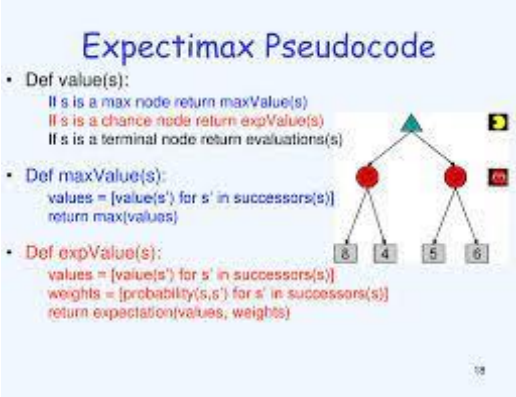
Expectimax

```
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (question 4)
    """
    action = None
    all_agent_num = 0

    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 ***"
        self.all_agent_num = gameState.getNumAgents()
        self.ExpectiMax(gameState, 0, 0)
        return self.action
        util.raiseNotDefined()

    def ExpectiMax(self, gameState, turn, depth):
        if gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState)
        agent_id = turn % self.all_agent_num
        if agent_id == 0:
            if depth >= self.depth:
                return self.evaluationFunction(gameState)
            else:
                return self.max_value(gameState, turn, depth)
        else:
            return self.exp_value(gameState, turn, depth)

    def max_value(self, gameState, turn, depth):
        v = -9999999
        legal_action = gameState.getLegalActions(0)
        for action in legal_action:
            state = gameState.generateSuccessor(0, action)
            maximum = self.ExpectiMax(state, turn + 1, depth + 1)
            if maximum > v:
                v = maximum
                if turn == 0:
                    self.action = action
        return v

```

```
    def exp_value(self, gameState, turn, depth):
        v = 0
        index = turn % self.all_agent_num
        legal_action = gameState.getLegalActions(index)
        for action in legal_action:
            state = gameState.generateSuccessor(index, action)
            v += ((1 / len(legal_action)) * self.ExpectiMax(state, turn
 + 1, depth))
        return v

```

تحلیل سوال چهارم :

- در این سوال به پیاده سازی Expectimax می پردازیم



Algorithm 2: Expectimax

Data: Game description, node u, player *player* to act as MAX
Result: The optimal move for *player* in node u's state
$(move, value) \leftarrow$ EXPECTIMAX-VALUE$(u, player)$
return *move*

function EXPECTIMAX-VALUE$(u, player)$:
  if u *is a chance node* **then**
    $expected \leftarrow 0$
    for $r \in moves(u)$ **do**
      $(move, value) \leftarrow$ EXPECTIMAX-VALUE$(result(u, r),$ *player*$)$
      $expected \leftarrow expected + P(r) \cdot value$
    end
    return $(NULL, expected)$
  else if $turn(u) = player$ **then**
    return MAX-VALUE$(u, player)$
  else
    // this is a MIN node
    return MIN-VALUE$(u, player)$
  end
end

چون فقط ExpectiMax داریم دیگر برای Min تابعی پیاده سازی نمیکنیم.تابع chance هم به نام expected_value نوشته شده است.

در اینجا گره های ما یا chance است یا maximizer .برای محاسبه ی chance از تقسیم کردن مجموع مقادیر بر منتسب به فرزندان بدست می اید.

- Evaluation Function

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

    DESCRIPTION: <write something here so we know what you did>
    """
    "*** YOUR CODE HERE ***"
    pacman_pos = currentGameState.getPacmanPosition()
    all_food = currentGameState.getFood().asList()
    score = currentGameState.getScore()
    ghost_states = currentGameState.getGhostStates()
    value = score
    for ghost in ghost_states:
        dist_ghost = util.manhattanDistance(pacman_pos,
ghost.configuration.getPosition())
        if dist_ghost != 0:
            if ghost.scaredTimer > 0:
                value += 100 / dist_ghost
            else:
                value -= 10 / dist_ghost

    # dist to closet food
    dist_food = []
    for food in all_food:
        dist_food.append(1 / util.manhattanDistance(food, pacman_pos))
    if len(dist_food) > 0:
        value += max(dist_food) * 10

    return value
    util.raiseNotDefined()

# Abbreviation
better = betterEvaluationFunction

```

تحلیل سوال پنج :

- در این سوال به پیاده سازی Evaluation Function می پردازیم

در این سوال تابع Evaluation Function را ارتقا دادیم تا تا نتیجه و امتیاز بیش تری کسب کنیم.

در این سوال روی روح اپشن های جدید می گذاریم و فرار از دست دشمن ها هم تغییر می کند.

در این سوال برای پیاده سازی از همه ی عوامل محیطی استفاده می کنیم .

نزدیک ترین فاصله غذا به pacman را محاسبه می کنیم فاصله ی روح های فعال ترسیده و فاصله ی ان ها به pacman قرار گرفته اند.

سپس به هر state یک priorty اختصاص داده و مقادیر را پیدا می کنیم .

- Conclusion

۱ با استفاده از فایل autograder.py و تست کردن برنامه در مجموع امتیاز ۲۵/۲۵ دریافت می شود.

۲ کدی که برای Reflex Agent نوشته شده توانست میانگین امتیازی معادل با عبارت زیر را کسب کند و تست های خود را پاس کند.

Average Score: 1241.5

۳ دو قسمت کد برای بخش های دو سه نوشتیم که عبارت بودند از Minimax و Alpha Beta Pruning با توجه به نتایجی که داشتیم دیدیم در مجموع Alpha Beta Pruning بسیار بهینه تر از Minimax عمل می کند.
و تعداد گره های کم تری مورد بررسی قرار گرفته و در نتیجه سرعت پیمایش محیط بازی و الگوریتم ما در نهایت بیش تر می شود.

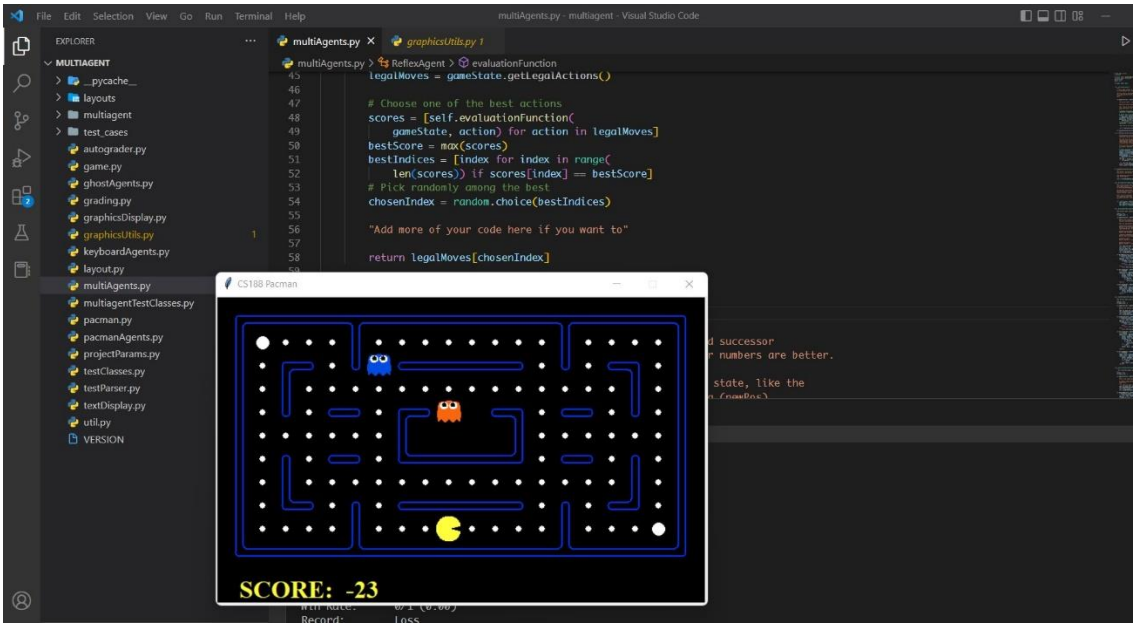
۴ در قسمت چهارم به پیاده سازی Expectimax داشتیم که به گونه ای عمل می کرد که روح ها به صورت تصادفی بازی می کردند و در محیط trappedClassic این ویژگی را داریم که این ویژگی با اجرا شدن در دو الگوریتم اولیه ی ما به طور کلی با شکست مواجه می شوند اما در این الگوریتم در حالت هایی که goast بهینه بازی نمی کند بازی را می بریم.

۵ پیاده سازی تابع Evaluation Function منجر به برد در تمامی حالات می شود. در این الگوریتم ما ارزش بیش تری به foods می دهیم و score هم ارزش بیش تری از goast دارد.

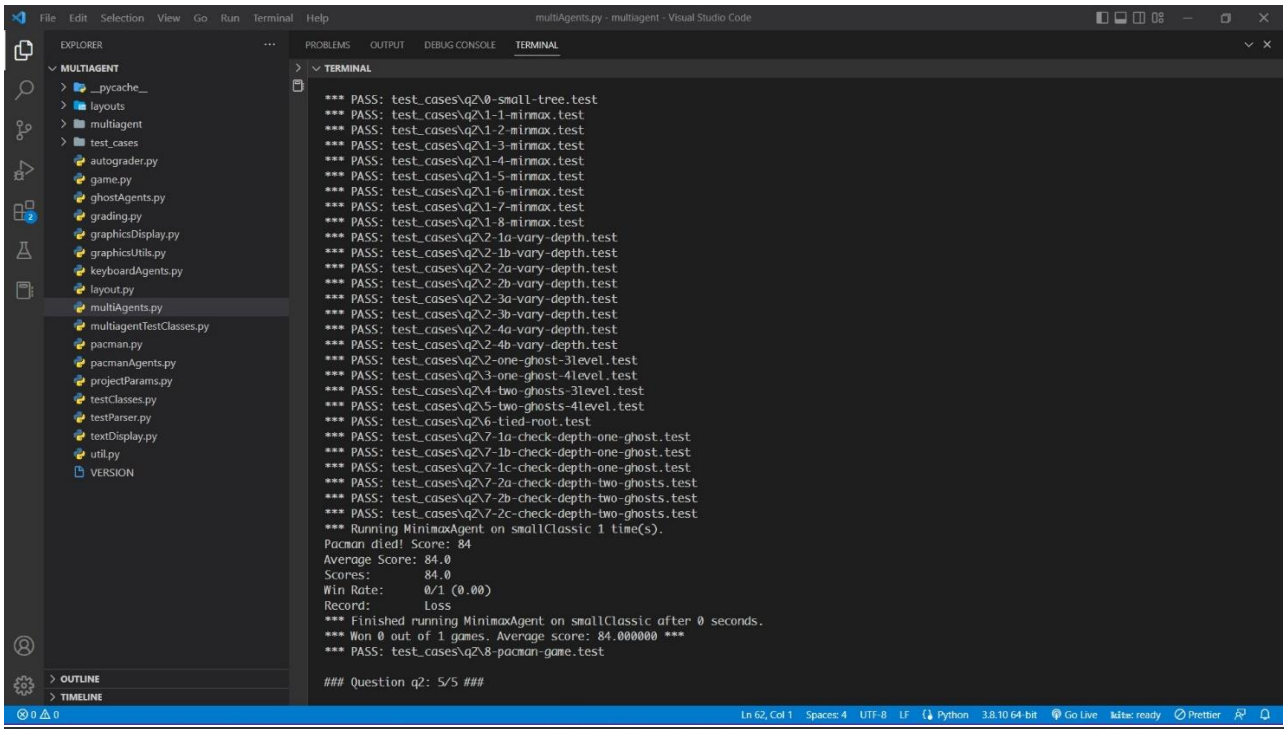
عکس از فضای گرافیکی :

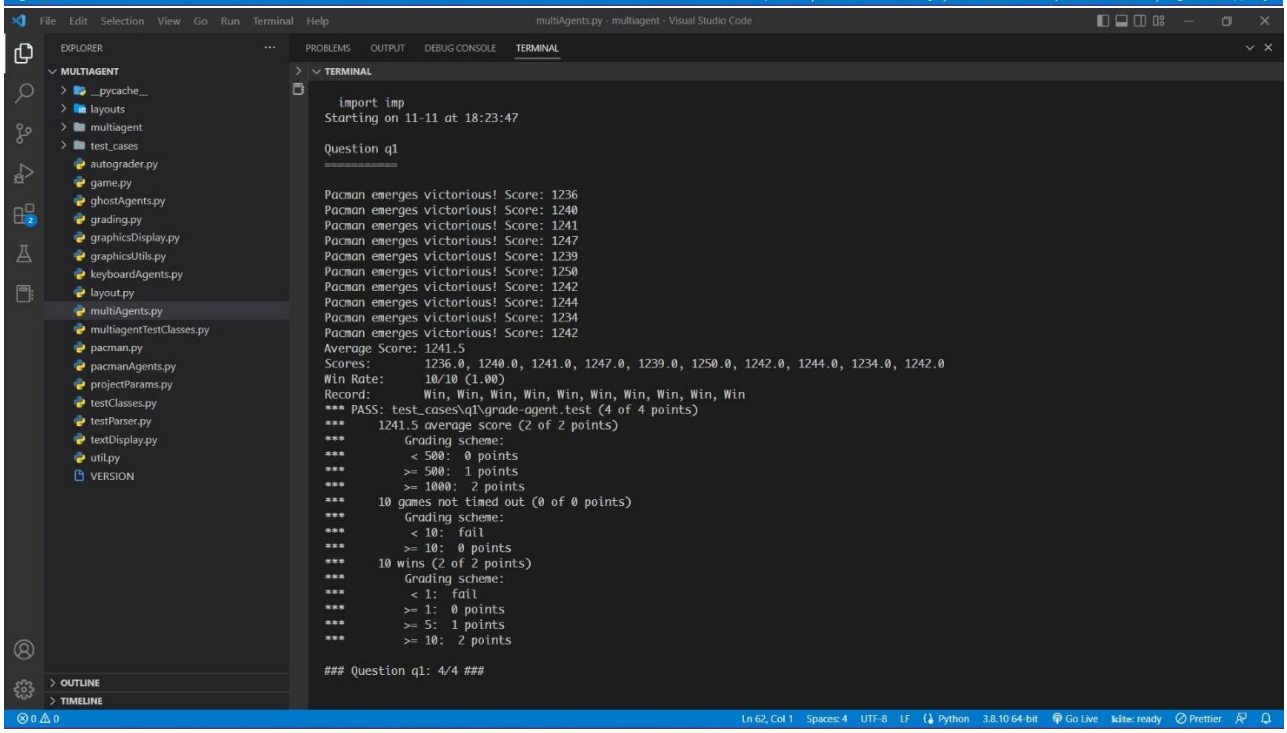
با اجرای دستورات

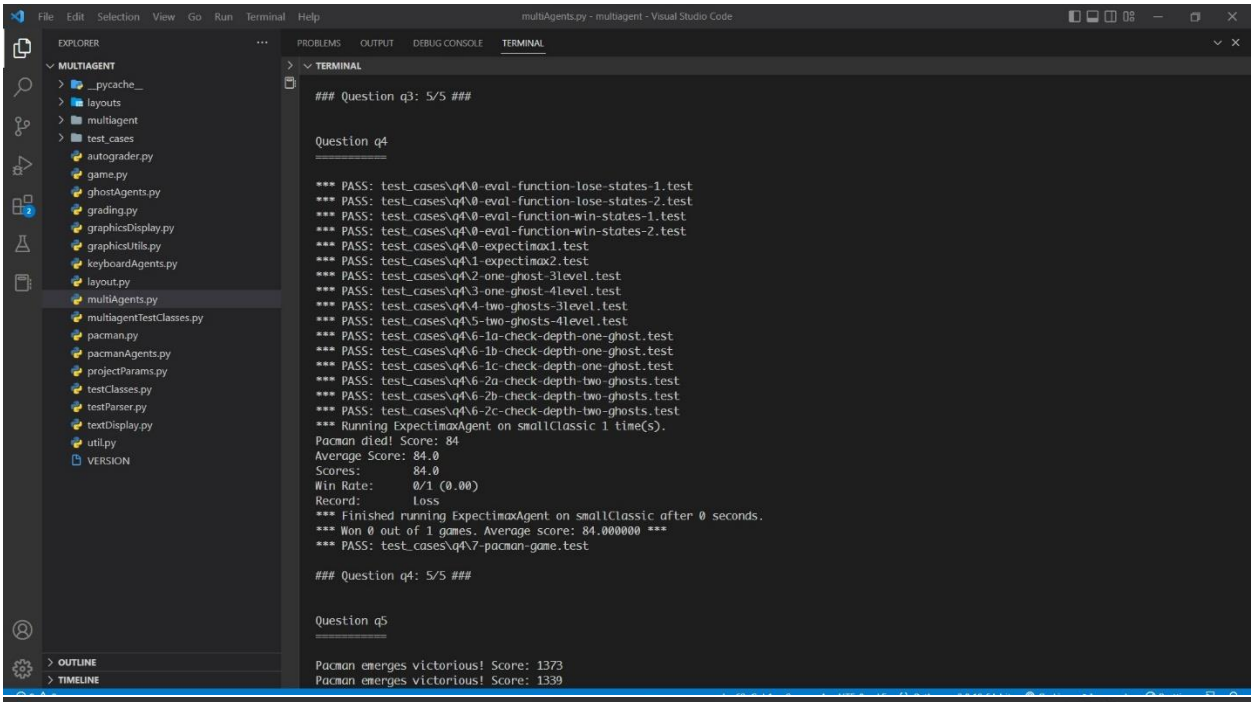
PS C:\Users\user\Desktop\multiagent> python pacman.py می توانیم محیط گرافیکی بازی را ببینیم.

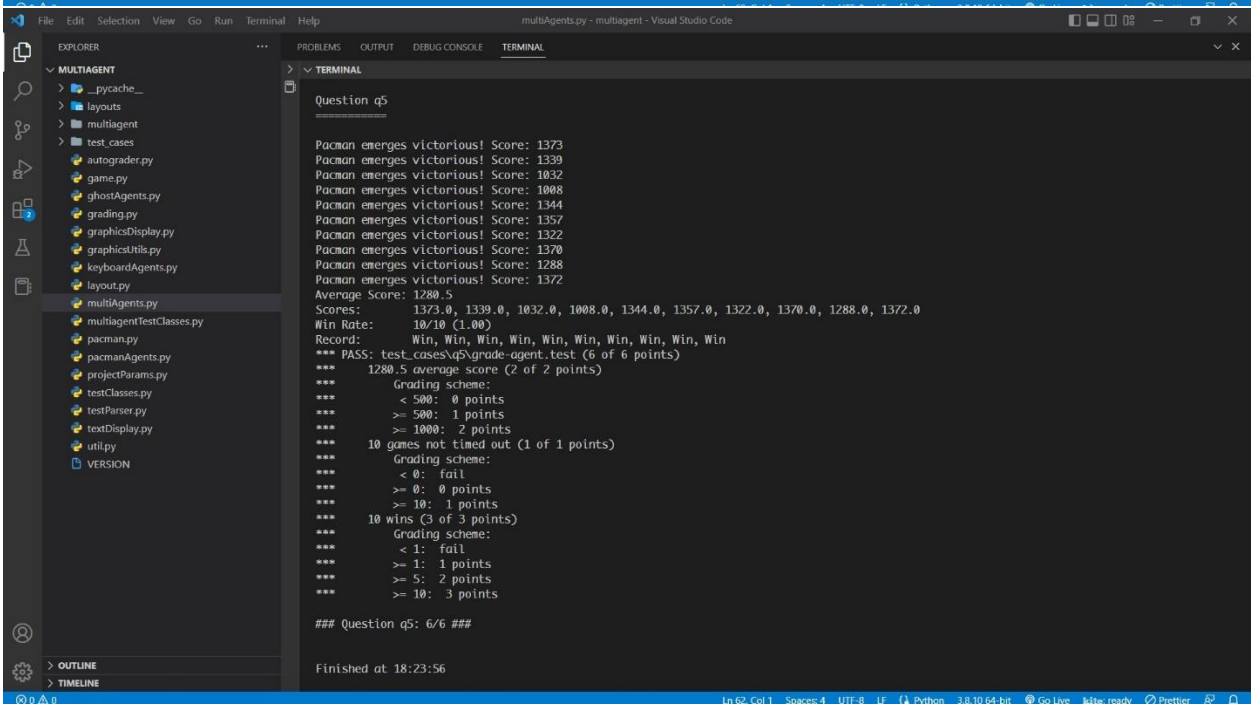

File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
multiAgents.py
graphicsUtils.py 1
multiAgents.py > ReflexAgent > evaluationFunction
legalMoves = gameState.getLegalActions()
# Choose one of the best actions
scores = [self.evaluationFunction(
gameState, action) for action in legalMoves]
bestScore = max(scores)
bestIndices = [index for index in range(
len(scores)) if scores[index] == bestScore]
# Pick randomly among the best
chosenIndex = random.choice(bestIndices)
"Add more of your code here if you want to"
return legalMoves[chosenIndex]
CS188 Pacman
SCORE: -23

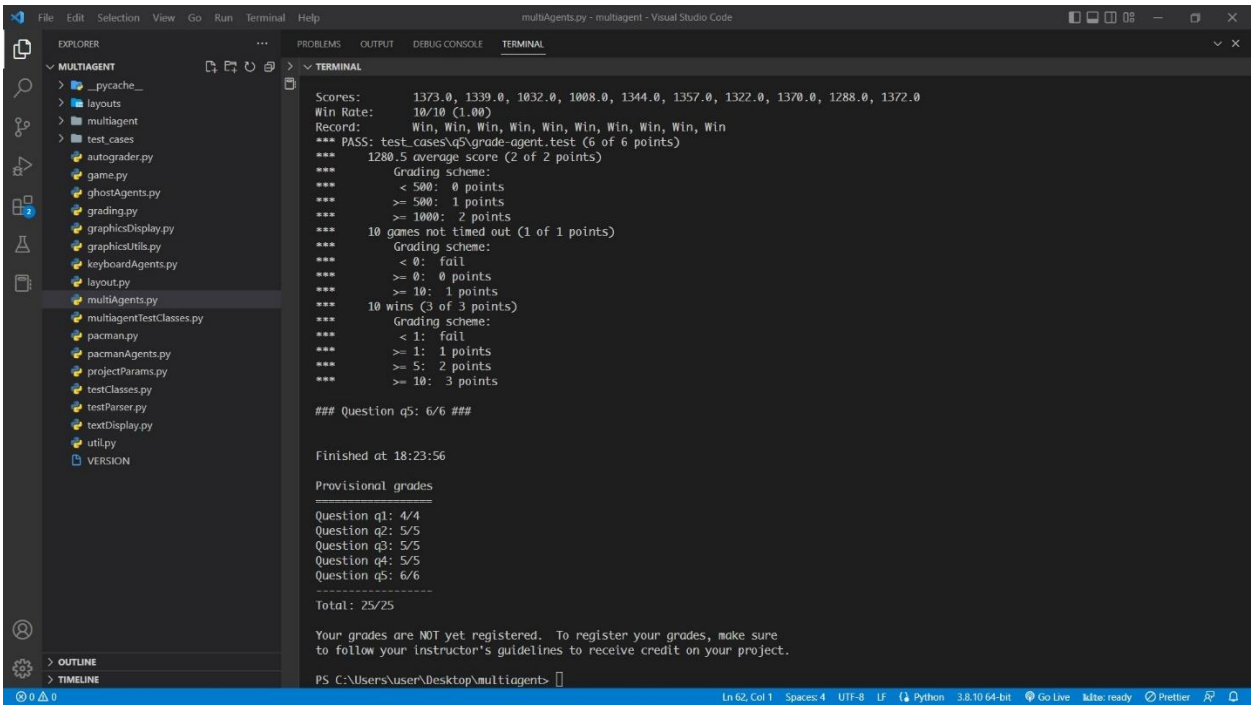
تست کردن :


File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
*** 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 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test
Question q2: 5/5
Ln 62, Col 1 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit Go Live kite: ready Prettier


File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
import imp
Starting on 11-11 at 18:23:47
Question q1
==========
Pacman emerges victorious! Score: 1236
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1241
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1250
Pacman emerges victorious! Score: 1242
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1234
Pacman emerges victorious! Score: 1242
Average Score: 1241.5
Scores: 1236.0, 1240.0, 1241.0, 1247.0, 1239.0, 1250.0, 1242.0, 1244.0, 1234.0, 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)
*** 1241.5 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
Ln 62, Col 1 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit Go Live kite: ready Prettier


File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
Question q3: 5/5
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
Question q5
==========
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1339


File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
Question q5
==========
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1339
Pacman emerges victorious! Score: 1032
Pacman emerges victorious! Score: 1008
Pacman emerges victorious! Score: 1344
Pacman emerges victorious! Score: 1357
Pacman emerges victorious! Score: 1322
Pacman emerges victorious! Score: 1370
Pacman emerges victorious! Score: 1288
Pacman emerges victorious! Score: 1372
Average Score: 1280.5
Scores: 1373.0, 1339.0, 1032.0, 1008.0, 1344.0, 1357.0, 1322.0, 1370.0, 1288.0, 1372.0
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)
*** 1280.5 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 18:23:56
Ln 62, Col 1 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit Go Live kite: ready Prettier


File Edit Selection View Go Run Terminal Help
multiAgents.py - multiagent - Visual Studio Code
EXPLORER
MULTIAGENT
__pycache__
layouts
multiagent
test_cases
autograder.py
game.py
ghostAgents.py
grading.py
graphicsDisplay.py
graphicsUtils.py
keyboardAgents.py
layout.py
multiAgents.py
multiagentTestClasses.py
pacman.py
pacmanAgents.py
projectParams.py
testClasses.py
testParser.py
textDisplay.py
util.py
VERSION
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
TERMINAL
Scores: 1373.0, 1339.0, 1032.0, 1008.0, 1344.0, 1357.0, 1322.0, 1370.0, 1288.0, 1372.0
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)
*** 1280.5 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 18:23:56
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
Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
PS C:\Users\user\Desktop\multiagent>
Ln 62, Col 1 Spaces: 4 UTF-8 LF Python 3.8.10 64-bit Go Live Ikite: ready Prettier