گزارش پروژه سوم

پروژه Pacman

درس: مبانی و کاربردهای هوش مصنوعی

استاد راهنما: دکتر حسین کارشناس نجف آبادی

اعضای گروه ۲ :

علی‌اکبر احراری- ۴۰۰۳۶۱۳۰۰۱

مهرآذین مرزوق- ۴۰۰۳۶۱۳۰۵۵

پاییز ۱۴۰۲

فهرست

گزارش کار الگوریتم

تابع scoreEvaluationFunction

```
def scoreEvaluationFunction(currentGameState: GameState):
    newPos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    numFood = currentGameState.getNumFood()
    newGhostPositions = currentGameState.getGhostPositions()
    score = currentGameState.getScore()

    # Run away from ghost
    if min([manhattanDistance(newPos, ghost) for ghost in newGhostPositions])
< 3:
        score -= 10

    # If ghosts are away and there is food, run to the food
    if all(manhattanDistance(newPos, ghost) > 5 for ghost in
newGhostPositions):
        distancesToFood = [manhattanDistance(newPos, food) for food in
newFood]
        if distancesToFood:
            score -= 1 / (numFood * min(distancesToFood) + 1)
        else:
            score += 1000

    return score
```

این تابع در محاسبه مقدار امتیاز عامل بر اساس ارزیابی‌های بدست آمده از محیط کاربرد دارد.

ابتدا مقادیر و وضعیت عوامل محیط را با استفاده از تابع CurrentGameState بدست می‌آوریم. پس از آن با استفاده از یک شرط، حداقل فاصله عامل از روح(ها) برای فرار از آن را در نظر می‌گیریم.

می‌توان مشاهده کرد که عامل در برخی از مراحل اجرای بازی تصمیم به توقف و ادامه ندادن حرکت به سوی مناطق دیگر بازی می‌گیرد. شرط دوم برای تشویق عامل به حرکت به سوی غذا درصورت دور بودن ارواح از آن می‌باشد.

کلاس MultiAgentSearchAgent

```
class MultiAgentSearchAgent(Agent):
    def __init__(self, evalFn="scoreEvaluationFunction", depth="2",
time_limit="6"):
        super().__init__()
        self.index = 0  # Pacman is always agent index 0
        self.evaluationFunction = util.lookup(evalFn, globals())
        self.depth = int(depth)
        self.time_limit = int(time_limit)
```

این کلاس در واقع کلاس پایه این بازی می‌باشد که پارامترهای مورد نیاز برای جستجو در بازی را مشخص و مقداردهی اولیه می‌کند.

کلاس AIAgent

این کلاس که از کلاس قبلی ارث می‌برد، حاوی پیاده‌سازی توابع مورد استفاده برای تصمیم‌گیری عامل برای انتخاب کنش مناسب است. در این بخش، الگوریتم minimax با استفاده از هرس alpha-beta برای تصمیم‌گیری مناسب پیاده‌سازی شده است.

تابع maxValue

```
def maxValue(self, gameState: GameState, depth, agentIndex, alpha, beta):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)
    value = -float("inf")
    for action in gameState.getLegalActions(agentIndex):
        value = max(value,
self.minValue(gameState.generateSuccessor(agentIndex, action),
                                          depth, agentIndex + 1, alpha, beta))
        if value > beta:
            return value
        alpha = max(alpha, value)
    return value
```

این تابع، منطق رساندن امتیاز عامل Pacman به بیشترین امتیاز ممکن (MAX) در درخت بازی را پیاده‌سازی می‌کند. بدین صورت که به صورت بازگشتی، کنش‌های ممکن برای عامل Pacman را بررسی کرده و کنش دارای بالاترین امتیاز را انتخاب می‌کند.

تابع minValue

```
def minValue(self, gameState: GameState, depth, agentIndex, alpha, beta):
    if gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)
    value = float("inf")
    for action in gameState.getLegalActions(agentIndex):
        if agentIndex == gameState.getNumAgents() - 1:
            value = min(value,
self.maxValue(gameState.generateSuccessor(agentIndex, action),
                                              depth + 1, 0, alpha, beta))
        else:
            value = min(value,
self.minValue(gameState.generateSuccessor(agentIndex, action),
                                              depth, agentIndex + 1, alpha,
beta))
        if value < alpha:
            return value
        beta = min(beta, value)
    return value
```

این تابع، منطق رساندن امتیاز عامل روح (Ghost) به کمترین امتیاز ممکن (MIN) در درخت بازی را پیاده‌سازی می‌کند. بدین صورت که به صورت بازگشتی، کنش‌های ممکن برای عامل روح را بررسی کرده و کنش دارای پایین‌ترین امتیاز را انتخاب می‌کند.

تابع getAction

```
def getAction(self, gameState: GameState):
    alpha = -float("inf")
    beta = float("inf")
    bestScore = -float("inf")
    bestActions = []  # This will hold all the best actions

    for action in gameState.getLegalActions(0):
        pacmanValue = self.maxValue(gameState.generateSuccessor(0, action),
0, 0, alpha, beta)
        print(f'action = {action}   value = {pacmanValue}')
        if action == Directions.STOP:
            pacmanValue -= 4
        if pacmanValue > bestScore:
            bestScore = pacmanValue
            bestActions = [action]  # Start a new list of best actions
        elif pacmanValue == bestScore:
            bestActions.append(action)  # Add action to the list of best
actions
        if bestScore > beta:
            return random.choice(bestActions)  # Choose randomly among the
best actions
        alpha = max(alpha, bestScore)

    return random.choice(bestActions)  # Choose randomly among the best
actions
```

این تابع به دلیل مقداردهی فرایند تصمیم‌گیری، نقش مهمی در این پروژه را ایفا می‌کند. وظیفه این، انتخاب بهترین کنش ممکن برای عامل Pacman در هر حالت با توجه به الگوریتم minimax می‌باشد.

در ابتدا، مقادیر bestScore، آلفا و بتا را به ترتیب برابر با منفی بی‌نهایت ،منفی بی‌نهایت و مثبت بی‌نهایت قرار می‌دهیم. یک لیست برای نگهداری بهترین کنش‌ها تشکیل می‌دهیم. در ادامه به حرکت در میان کنش های ممکن می‌پردازیم. برای هر حرکت، maxValue را محاسبه و بهترین کنش را رسد می‌کنیم. همچنین یک شرط برای ترغیب عامل به ادامه حرکت خود و جلوگیری از ایجاد وقفه طولانی تعریف کرده ایم.

در ادامه بهترین کنش ممکن را با مقایسه بهترین امتیاز در حال حاضر و امتیاز فعلی Pacman به دست می‌آوریم و در صورت بیشتر بودن امتیاز Pacman، آن کنش را به لیست اضافه می‌کنیم.

با استفاده از هرس آلفا و بتا می‌توان در زمان در انتخاب کنش مناسب صرفه‌جویی کرد. در این شرط، اگر بیشترین امتیاز بزرگتر از بتا بود، این شاخه را هرس می‌کنیم و یک کنش از لیست bestActions به صورت تصادفی برای حرکت بعدی انتخاب می‌کنیم.

منابع مورد استفاده

https://www.youtube.com/watch?v=3pU-Hrz_xy4

کتابخانه‌های مورد استفاده

```
from util import manhattanDistance
from game import Directions
import random
import util
from game import Agent
from pacman import GameState
```

util: برخی توابع پرکاربرد مانند تابع manhattanDistance که در پیدا کردن فاصله کاربرد دارد را در اختیارمان می‌گذارد.

Random: استفاده از متغیرهای تصادفی.

Game: استفاده از عوامل مختلف بازی مانند عامل و جهات ممکن حرکت آن.

Pacman: مورد استفاده برای شناسایی وضعیت بازی.