دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

پروژه :

Pacman ، پیاده سازی الگوریتم MiniMax

اعضای گروه :

حمید مهران فر (۴۰۰۳۶۱۳۰۵۸)

رادمهر آقاخانی (۴۰۰۳۶۶۳۰۰۲)

علی کثیری (۴۰۰۳۶۱۳۰۵۱)

استاد : حسین کارشناس

گروه ۱۰

زمستان ۱۴۰۲

تابع getPossibleActions :

این تابع حرکت های مجاز عامل را برمیگرداند . تنها تفاوت این تابع با تابع getLegalActions کلاس GameState این است که حرکت stop را از مجموعه حرکات ان حذف می کند .

```
def getPossibleActions(gameState, player):
    legalAction = gameState.getLegalActions(player)
    if Directions.STOP in legalAction:
        legalAction.remove(Directions.STOP)
    return legalAction
```

تابع scoreEvaluationFunction :

این تابع امتیاز گره های برگ را بر میگرداند . ابتدا مقادیر مورد نیاز برای محاسبه امتیاز را بدست می اوریم .

```
newPos = currentGameState.getPacmanPosition()
newFood = currentGameState.getFood()
newGhostStates = currentGameState.getGhostStates()
```

سپس ثابت ها را تعریف می کنیم که برای امتیاز دهی استفاده می شوند . ثابت WEIGHT_FOOD برای وزن دهی به فاصله تا غذا ، ثابت WEIGHT_GHOST برای وزن دهی به فاصله تا روح و WEIGHT_SCARED_GHOST برای وزن دهی به فاصله تا روح ترسیده است . مقادیر ثابت ها به ترتیب برابر با ۵ ، ۵- و ۵۰ است .

```
# Consts
INF = 100000000.0  # Infinite value for being dead
WEIGHT_FOOD = 5.0  # Food base value
WEIGHT_GHOST = -5.0  # Ghost base value
WEIGHT_SCARED_GHOST = 50.0  # Scared ghost base value
```

سپس با استفاده از تابع manhattanDistance از ماژول util قدرمطلق فاصله ی غذا ها تا عامل را بدست می اوریم . برای امتیاز دهی ، ثابت WEIGHT_FOOD بر کمترین فاصله تقسیم کرده و با

امتیاز جمع شده تا این نقطه از بازی جمع می کنیم . اگر هم فاصله ای وجود نداشته باشد ، ثابت را با امتیاز جمع شده ، جمع می کنیم .

```
score = currentGameState.getScore()

# Evaluate the distance to the closest food
distancesToFoodList = [util.manhattanDistance(newPos, foodPos) for foodPos in newFood.asList()]
if len(distancesToFoodList) > 0:
    score += WEIGHT_FOOD / min(distancesToFoodList)
else:
    score += WEIGHT_FOOD
```

سپس قدرمطلق فاصله ی عامل با روح ها را حساب می کنیم . اگر فاصله صفر بود ، یعنی عامل خورده شده است و به همین خاطر امتیاز منفی بینهایت را بر میگرداند . اگر روح در حال ترسیدن نبود ، ثابت WEIGHT_GHOST که یک عدد منفی است بر فاصله تقسیم شده و با امتیاز بدست امده جمع می شود . همچنین اگر روح در حال ترسیدن باشد ، ثابت WEIGHT_SCARED_GHOST بر فاصله تقسیم شده و با امتیاز بدست امده جمع شود . این کار باعث می شود که عامل به دنبال روح درحال ترس برود.

```
# Evaluate the distance to ghosts
for ghost in newGhostStates:
    distance = manhattanDistance(newPos, ghost.getPosition())
    if distance > 0:
        if ghost.scaredTimer > 0:  # If scared, add points
            score += WEIGHT_SCARED_GHOST / distance
        else:  # If not, decrease points
            score += WEIGHT_GHOST / distance
    else:
        return -INF  # Pacman is dead at this point
# print(score)
return score
```

کلاس MiniMaxAgent :

تابع minimax : در این تابع الگوریتم minimax پیاده سازی شده است . ورودی های تابع عمق و محیط بازی و بازیکن است . عمق درواقع نشان دهنده ی میزان پیشروی در الگوریتم است . بازیکن هم برای بررسی گره های min و max است که اگر صفر باشد ، یعنی عامل و اگر غیر صفر باشد یعنی روح های بازی . اگر بازی تمام شده باشد یا عمق جستجو برابر با عمق مشخص شده باشد ، evaluationFunction فراخوانی می شود. اگر جستجو در گره ی max باشد ، تابع getMax و اگر در گره ی Min باشد ، گره ی getMin فراخوانی می شود .

```
def minimax(self, depth, gameState, player):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)
    if player == 0:
        return self.getMax(depth, gameState, player)
    else:
        return self.getMin(depth, gameState, player)
```

تابع getMax : این تابع بیشترین مقدار گره های فرزند را برمیگرداند . به این صورت که جستجو را به صورت بازگشتی (عمقی) انجام داده و سپس بیشترین مقدار را برمیگرداند .

تابع getMin : این تابع هم کمترین مقدار گره های فرزند را برمیگرداند . اگر بازیکن برابر با اخرین روح بررسی نشده باشد ، این تابع جستجو را روی عامل انجام می دهد . همچنین عمق یک واحد زیاد می شود .

تابع getAction : این تابع با استفاده از الگوریتم minimax ارزش گره های فرزند را بدست می اورد و سپس بهترین انرا انتخاب می کند . همچنین اگر چندین ارزش ، بهترین مقادیر باشند ، به صدرت رندوم انتخاب می شود .

کلاس AIAgent :

تابع alphaBeta : در این تابع الگوریتم هرس آلفا – بتا پیاده سازی شده است . این تابع پنج ورودی می گیرد که سه مورد ان قبلا توضیح داده شد . دو ورودی دیگر متغیر های آلفا و بتا هستند که در ابتدای کار برابر با بینهایت هستند . این دو ورودی برای هرس درخت مورد استفاده قرار می گیرند . اگر هم بازی تمام شود یا عمق جستجو برابر با عمق مشخص شده باشد ، evaluationFunction فراخوانی می شود . همچنین اگر جستجو در گره ی max باشد ، تابع alphaPart و اگر در گره ی min باشد ، تابع betaPart فراخوانی می شود .

```
def alphaBeta(self, depth, gameState, player, alpha, beta):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)
    if player == 0:
        return self.alphaPart(depth, gameState, player, alpha, beta)
    else:
        return self.betaPart(depth, gameState, player, alpha, beta)
```

تابع alphaPart : این تابع بیشترین مقدار گره های فرزند را بر میگرداند . به این صورت که هر بار بیشترین مقدار گره های بررسی شده را با مقدار بتا مقایسه می شود و اگر بزرگتر بود ، درخت هرس میشود . در غیر این صورت مقدار الفا برابر با بیشترین مقدار الفا و گره های بررسی شده می شود . این کار باعث کمتر شدن درخت جستجو می شود .

```
def alphaPart(self, depth, gameState, player, alpha, beta):
    maxValue = float('-inf')
    legalActions = getPossibleActions(gameState, player)
    for action in legalActions:
        maxValue = max(maxValue, self.alphaBeta(depth, gameState.generateSuccessor(player, action), 1, alpha, beta))
        if maxValue >= beta:
            break
        alpha = max(alpha, maxValue)
    return maxValue
```

تابع betaPart : این تابع کمترین مقدار گره های فرزند را بر میگرداند . به این صورت که هر بار کمترین مقدار گره های بررسی شده با الفا مقایسه می شود و اگر کمتر بود ، بقیه ی گره های فرزند بررسی نمی شوند . همچنین بتا برابر با کمترین مقدار گره های بررسی شده و خودش می شود . اگر هم درحال بررسی اخرین روح بررسی نشده باشد ، جستجوی بعدی روی عامل انجام می شود و عمق هم یک واحد زیاد می شود .

```
def betaPart(self, depth, gameState, player, alpha, beta):
    nextPlayer = player + 1
    if player == gameState.getNumAgents() - 1:
        nextPlayer = 0
    if nextPlayer == 0:
        depth += 1

    minValue = float('inf')
    legalActions = getPossibleActions(gameState, player)
    for action in legalActions:
        minValue = min(minValue, self.alphaBeta(depth, gameState.generateSuccessor(player, action), nextPlayer, alpha, beta))
        if minValue <= alpha:
            break
        beta = min(beta, minValue)
    return minValue
```

تابع getAction : این تابع با استفاده از الگوریتم هرس آلفا – بتا ارزش گره های فرزند را بدست می اورد و سپس بهترین انرا انتخاب می کند . همچنین اگر چندین ارزش ، بهترین مقادیر باشند ، به صدرت رندوم انتخاب می شود .

```
def getAction(self, gameState: GameState):
    alpha = float('-inf')
    beta = float('inf')
    legalActions = getPossibleActions(gameState, 0)
    bestAction = []
    for action in legalActions:
        bestAction.append(self.alphaBeta(0, gameState.generateSuccessor(0, action), 1, alpha, beta))
    choosen = np.argmax(bestAction)
    max_indices = [index for index in range(len(bestAction)) if bestAction[index] == bestAction[choosen]]
    chosenIndex = random.choice(max_indices)
    return legalActions[chosenIndex]
```

کتابخوانه های مورد استفاده :

random , numpy , util , Direction , Agent , GameState

منابع :

https://web.stanford.edu/class/archive/cs/cs221/cs221.1196/assignments/pacman/index.html

https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

https://www.aparat.com/v/1bOre

https://fa.wikipedia.org/wiki/%D9%87%D8%B1%D8%B3_%D8%A2%D9%84%D9%81%D8%A7_%D8%A8%D8%AA%D8%A7#:~:text=%D9%87%D8%B1%D8%B3%20%D8%A2%D9%84%D9%81%D8%A7%20%D8%A8%D8%AA%D8%A7%20%D8%A7%D9%84%DA%AF%D9%88%D8%B1%DB%8C%D8%AA%D9%85%DB%8C%20%D8%A7%D8%B3%D8%AA,%D8%AF%D8%B1%20%D8%B2%D9%85%D8%A7%D9%86%DB%8C%20%DA%A9%D9%85%E2%80%8C%D8%AA%D8%B1%20%D8%B5%D9%88%D8%B1%D8%AA%20%D9%85%DB%8C%E2%80%8C%DA%AF%DB%8C%D8%B1%D8%AF.