به نام خدا

امیر بهنام -۹۸۳۱۱۳۳- گزارش پروژه دوم

سوال اول:

برای حل این سوال همانطور که در راهنمایی آمده است، برای ساخت یک تابع ارزیابی مناسب، به جای استفاده از مقادیر اطلاعات، از رابطه ی بین آنها استفاده شده است که در این سوال، ما از رابطه پکمن و غذا با استفاده از

manhattanDistance فاصله را یافته سپس کمترین فاصله با غذا را به عنوان میزان ارزیابی انتخاب کرده ایم.

```
if action == Directions.STOP:
   return float("-inf")
```

در قطعه کد بالا، در صورتی که حرکت مورد نظر از نوع استپ باشد، عدد منفی بزرگی را برمیگردانیم، این کار باعث میشود، این حرکت دارای کمترین اولویت باشد.

```
for ghostState in newGhostStates:

   if (
      ghostState.getPosition() == newPos
      and ghostState.scaredTimer == 0
   ):
      return float("-inf")
```

در قطعه کد فوق، بررسی میکنیم آیا حرکت جدید باعث باخت پکمن میشود یا خیر، باخت در صورتی اتفاق می افتد که پکمن به روحی برخورد کند که scaredنباشد.

```
distance = []

for food in foodList:
   distance.append(manhattanDistance(food, newPos))
```

در نهایت در قطعه کد فوق، فاصله تمامی غذا ها محاسبه میشود.

```
return -min(distance)
```

سپس در عبارت بالا، پس از انتخاب مینیمم فاصله غذا، مقدار آن به صورت منفی برگشت داده میشود. این عدد به این دلیل منفی است چون این مقدار، مینیمم مقدار هزینه ای است که پکمن برای رسیدن به غذا باید طی کند.

سوال دوم:

در این سوال، بیشتر سعی در پیاده سازی الگوریتم مین ماکس بوده است .یک لایه ماکس برای پکمن و به تعداد روح ها لایه مین ایجاد میشود .همیشه اندیس 0 برای پکمن و سایر اندیس ها برای روح ها است که طبق الگوریتم، در هر عمق اندیس بعدی بررسی میشود و

در صورتی که همه اندیس ها بررسی شد به عمق بعدی میرویم .در این مرحله پکمن قصد دارد امتیاز بیشتری کسب کند و اگر در مرحله ای به این نتیجه برسید که باخت، نسبت به ادامه بازی، امتیاز بیشتری برای پکمن کسب میکند، سعی میکند سریع تر ببازد.

```
def getAction(self, gameState):
    val = self.getValue(gameState, 0, 0)
    return val[0]
```

در قطعه کد بالا، تابع اصلی برای پیدا کردن حرکت مورد نظر آورده شده است. این تابع مقدار gamestate، و همچنین مقدار 0 برای اندیس، و مقدار 0 برای عمق را به تابع getValue ارسال می کند. همچنین در نهایت مقدار اندیس صفر متغیر valکه حرکت را مشخص میکند را برمیگرداند. تابع getvalueبه شرح زیر است:

```
def getValue(self, gameState, index, depth):
    if index >= gameState.getNumAgents():
        index = 0
        depth += 1
```

در تکه کد بالا، همانطور که توضیح داده شد، بررسی میشود آیا همه اندیس ها بررسی شده اند یا خیر، اگر بررسی شده باشند شرط بالا صحیح بوده و اندیس صفر میشود و جستجو به عمق بعدی می رود

```
    if depth == self.depth:
        return self.evaluationFunction(gameState)
```

در صورتی که عمق جستجو شده، به میزان عمق سوال رسیده باشد، مقدار فوق بازگشت داده میشود. لازم به ذکر است تابع فوق داخل کلاس های والد این کلاس وجود دارند، و یک زوج که شامل حرکت و مقدار آن است را برمیگرداند.

```
    if index == 0:
        return self.maxValue(gameState, index, depth)
```

در صورتی که اندیس صفر باشد، ماکس را فراخوانی میکنیم

```
    else:
            return self.minValue(gameState, index, depth)
```

در صورتی که اندیس صفر نباشد مقدار مین را برای روح فراخوانی میکنیم

در لایه مین،

```
def minValue(self, gameState, index, depth):
    li = [];
```

ابتدا یک لیست خالی را در نظر میگیریم

```
    allactions = gameState.getLegalActions(index)
```

سپس تمامی حرکت های ممکن را مشخص میکنیم

```
    if len(allactions) == 0:
```

```
return self.evaluationFunction(gameState)
```
در صورتی که هیچ حرکتی موجود نباشد، مقدار فوق را برمیگردانیم

```
for action in allactions:
```

اکنون تمامی حرکت ها را یکی یکی بررسی میکنیم

```
if action == Directions.STOP:
    continue
```
اگر حرکتی، استپ باشد، نیاز به بررسی نداشته و از آن عبور میکنیم

```
value = self.getValue(gameState.generateSuccessor(index, action), index + 1, depth)
```

مقدار متناظر را از تابع getvalueدریافت میکینم

در این قسمت توجه نمایید که اندیس، یکی افزایش یافته است و این به این معنی است که ما در حال بررسی اندیس بعدی هستیم.

```
if type(value) is tuple:
    value = value[1]
```

اگر نوع متغیر خروجی زوج باشد(ممکن است در برخی از تست ها، حرکت برگشت داده شده از ساکسسور، زوج نباشد چنانچه اگر این مورد در نظر نگرفته شده باشد، ممکن است مانند پروژه قبلی در برخی از موارد به مشکل برسیم) مقدار عددی آن را در متغیر v ذخیره میکنیم

```
li.append((action, value))
```
سپس نوع حرکت و مقدار آن را در لیست ذخیره میکنیم

```
return min(li, key = lambda x:x[1])
```

در نهایت ماکزیمم لیست را بر اساس پارامتر دوم که مقدار آن است را برمیگردانیم

```
return max(li, key = lambda x:x[1])
```
در لایه مینیمم، تنها تفاوت در نوع خروجی است که ما ماکزیمم لیست را برمیگردانیم.

سوال سوم:

در این سوال سعی میکنیم با برش هایی با نام هرس آلفا بتا، میزان پیمایش درخت را بهینه کنیم و شاخه هایی که نتیجه ی ضعیف تری را دارند، پیمایش نکنیم .در این سوال، مقادیر آلفا و بتا، در توابع ماکزیمم و مینیمم، به روز رسانی میشوند.

در این سوال تابع getvalueتفاوتی با سوال قبل ندارد. اما تغییرات مهمی در لایه مین و ماکس وجود دارد. در توابع مین و ماکس متغیری به نام lastتعریف شده است، که در حالت اولیه، یک زوج است که شامل یک حرکت استاپ با مقدار منفی بینهایت است. در هر مرحله از بررسی ها، در صورت یافتن مقدار بهتر، این متغییر تغییر کرده و مقدار بهینه را در آن ذخیره میکنیم.

```
if value <= last[1]:
    last = (action, value)

if last[1] < alpha:
    return last
beta = min(beta, last[1])
```

در قسمت مین، در صورتی که مقدار getvalue از مقدار آلفا کمتر باشد، این مقدار و حرکت در lastذخیره میشود. در غیر این صورت مقدار بتا برابر با این مقدار میشود در صورتی که از بتا کمتر باشد

در قسمت ماکس نیز، برخلاف قسمت بالا، در صورتیکه getvalue از بتا کمتر باشد، lastبرگردانده میشود. و در آلفا نیز برابر با ماکزیمم آلفا و lastمیشود.

سوال چهارم:

در این سوال تابع مینیمم سازی تغییر میکند .بر این اساس، روح ها میتوانند حرکتشان را با استفاده از تابع احتمال یکنواخت انتخاب کنند .تابع یکنواخت، مقداری برابر با 1 تقسیم بر تعداد روح ها را برمیگرداند . در این سوال، تفاوت ساختاری با سوال دوم وجود ندارد. جز در لایه مینیمم سازی.

```
e = 1.0/len(allactions)
```

که در قطعه کد بالا، نحوه محاسبه این مقدار آورده شده است.

سوال پنجم:

در این سوال، ما بر اساس برخی پارامترها مانند موقعیت روح ها، موقعیت غذا ها و موقعیت کپسول ها، مجموعی از حالات را بررسی و ترکیبی از آنها را به عنوان خروجی برمیگرداند .برخی از پارامترها مانند ضرایب امتیاز فعلی و .. به صورت آزمایش و خطا انتخاب شده است.

```
pos = currentGameState.getPacmanPosition()
```

در خط بالا موقعیت پکمن دریافت شده است.

```
manhattean_ghost = [ manhattanDistance(x.getPosition(), pos) for x in
currentGameState.getGhostStates()]
```

سپس لیستی از موقعیت روح ها ساخته شده است.

```
manhattean_food = [ manhattanDistance(x, pos) for x in currentGameState.getFood().asList()]
if len(manhattean_food) == 0:
    manhattean_food .append(1)
```

همچنین لیستی از موقعیت غذا ها نسبت به پکمن تهیه شده است.

```
    s = -min(manhattean_food) +min(manhattean_ghost)/1000 +currentGameState.getScore() -
50*len(currentGameState.getCapsules())
    return  s
```

در نهایت نیز مقدار متناظر برگردانده میشود.