گزارش پروژه مونوپولی

نام و نام خانوادگی:امیرحسین رستمی

دانشگاه:صنعتی امیرکبیر

نام استاد:دکتر مهدی قطعی

مقدمه

مونوپولی یک بازی تخته ای است در این بازی هر شرکت کننده به کمک تاس در خانه های مختلف حرکت می کند و املاکی که در زمین بازی وجود دارد را خریداری می کند و تجارت خود را با خرید هتل گسترش می دهد. بازیکنان از حریف خود اجاره دریافت می کنند تا آن ها را به سمت ورشکستگی بکشانند.در این پروژه این بازی شانسی را با کمک هوش مصنوعی پیاده سازی کرده ایم که در ان یک کاربر میتواند با یک عامل هوشمند بازی کند .

1-1 قوانین بازی

- این بازی دو بازیکن دارد که بازیکن اول کاربر(انسان) و بازیکن دوم نیز یک ربات(عامل هوشمند) است.
- محیط بازی شامل تعدادی مکان است که هر مکان دارای ویژگی خاصی است.
- در ابتدای بازی هر دو بازیکن در مکان Start قرار میگیرند و با انداختن تاس ، بر روی خانه ها جابجا میشوند.
- در ابتدای بازی برای هر بازیکن مبلغ 1000 دلار به عنوان پول نقد داده میشود.
- عملیات قابل انجام برای هر خانه شامل:1)Buy 2)Rent 3)Sell 4)DoNothing میباشد.
- هر بازیکن زمانی عمل Buy را میتواند انجام دهد که میزان پول نقدش بیشتر از قیمت آن خانه باشد.

- هر بازیکن اگر به خانه ای برسد که صاحب آن خانه ،بازیکن حریف باشد، انگاه باید برای بازیکن حریف اجاره بپردازد
- هر بازیکن اگر به خانه ای برسد که صاحب خانه خودش باشد، میتواند آن خانه را بفروشد و مبلغ فروش آن به موجودی نقدش اضافه میشود.
- برای هر بازیکن در هر خانه یک عمل DoNothing وجود دارد که در ان هیچ کاری انجام نمیدهد.(البته اگر صاحب خانه،بازیکن حریف باشد انگاه بازیکن نمیتواند عمل DoNothing را انجام دهد و باید حتما اجاره بپردازد)
- اگر بازیکن به خانه ای برسد که جریمه داشته باشد(مانند زندان یا اداره مالیات)،باید حتما مبلغ جریمه را بپردازد.
- زمانی بازی به پایان میرسد یکی از بازیکن ها ورشکست شوند یعنی پول نقد آن ها صفر شود.

2) پیاده سازی

برای پیاده سازی بازی از زبان پایتون و سبک برنامه نویسی شی گرا استفاده شده است.

2-1) کلاس ها

Property : هر خانه در محیط بازی یک کلاس Property است.

```python
class Property:
    def __init__(self,position:int,name:str,owner:int,hastax:bool,price:int,rentprice:int):
        self.HasTax:bool=hastax
        self.Name:str=name
        self.Price:int=price
        self.RentPrice:int=rentprice
        self.Position:int=position
        self.Owner=owner  # agentId || null
```

- Name : نام خانه
- Price :قیمت خرید خانه
- RentPrice : قیمت اجاره خانه
- Position : جایگاه هر خانه در محیط بازی
- Owner : شناسه بازیکنی که صاحب خانه است.
- HasTax : اگر خانه دارای جریمه باشد مقدار True و اگر دارای جریمه نباشد مقدار False میگیرد.
-

Properties : لیستی از کلاس Property است که اطلاعات اولیه همه خانه ها در آن قرار میگیرد.

```python
Properties:list[Property]=[
    Property(0,"Start",None,False,0,0),
    Property(1,"salmas",None,False,200,100),
    Property(2,"tabriz",None,False,200,100),
    Property(3,"Airport",None,False,300,100),
    Property(4,"Electric Company",None,False,400,100),
    Property(5,"prison",None,True,200,0),
    Property(6,"shiraz",None,False,200,100),
    Property(7,"tehran",None,False,200,100),
    Property(8,"mashhad",None,False,200,100),
    Property(9,"zanjan",None,False,200,100),
]
```

Agent : هر نمونه از Agent ، یک بازیکن است که میتواند از نوع انسان یا ربات باشد.

```python
class  Agent:
    def __init__(self,type:str,id:int,name:str,position:int,cash:int,properties:list[Property]=[]):
        self.Type=type #human or robot
        self.Id=id
        self.Name=name
        self.cash=cash
        self.Properties=properties
        self.Position=position
```

- Type : نوع Agent را مشخص میکند که مقادیر human یا robot را میتواند بگیرد.

- Id :شناسه هر بازیکن
- Name : نام هر بازیکن
- Cash :مقدار پول نقد بازیکن
- Properties:خانه هایی که صاحبش است
- Position :موقعیت فعلی بازیکن در جریان بازی

State : این کلاس وضعیت فعلی بازی را در خود نگه میدارد.

```
class State:
    def __init__(self,agent1:Agent,agent2:Agent,properties:list[Property]):
        self.Agent1=agent1
        self.Agent2=agent2
        self.Properties=properties


    def printState(self):
        print("*********state*********")
        print("agent1.position: ",self.Agent1.Position)
        print("agent1.cash: ",self.Agent1.cash)
        agent1props=[]
        for prop in self.Agent1.Properties:
            agent1props.append((prop.Name,prop.Position))
        print("agent1.propeties: ",agent1props)

        #agent2
        print("agent2.position: ",self.Agent2.Position)
        print("agent2.cash: ",self.Agent2.cash)
        agent2props=[]
        for prop in self.Agent2.Properties:
            agent2props.append((prop.Name,prop.Position))
        print("agent2.propeties: ",agent2props)
```

- Agent1 :این پراپرتی اطلاعات بازیکن اول را که از نوع انسان است در خود نگه میدارد.
- Agent2 : این پراپرتی اطلاعات بازیکن دوم را که از نوع ربات است در خود نگه میدارد.

- Properties : اطلاعات خانه های بازی در State فعلی بازی را نشان میدهد.
- PrintState() : وضعیت بازی را بر اساس state بازی در فرمت خاصی چاپ میکند.

- IsTerminal(self) : بررسی میکند که آیا وضعیت فعلی بازی ، وضعیت پایانی است یا خیر.به عبارت دیگر این متد بررسی میکند که اگر میزان پول نقد هر یک از بازیکنان در وضعیت فعلی کمتر یا مساوی صفر باشد، انگاه مقدار True را برمی گرداند.

```
def IsTerminal(self):
    if self.Agent1.cash<=0 or self.Agent2.cash <=0 :
        return True
```

- GetBuyState(self,player) : با توجه به بازیکن و حالت فعلی بازی، این متد عمل Buy را برای بازیکن مورد نظر انجام میدهد و State جدید را بر اساس عمل انجام شده برمیگرداند.

```
def GetBuyState(self,player:str):
    agent1=copy.deepcopy(self.Agent1)
    agent2=copy.deepcopy(self.Agent2)
    properties=copy.deepcopy(self.Properties)
    if player=="human":
        currentproperty=copy.deepcopy(properties[agent1.Position])
        agent1.cash=agent1.cash - currentproperty.Price
        agent1.Properties.append(currentproperty)
        properties[agent1.Position].Owner=agent1.Id

    elif player=="max":
        currentproperty=copy.deepcopy(properties[agent2.Position])
        agent2.cash=agent2.cash - currentproperty.Price
        agent2.Properties.append(currentproperty)
        properties[agent2.Position].Owner=agent2.Id


    return State(agent1,agent2,properties)
```

- GetSellState() : با توجه به بازیکن و حالت فعلی بازی، این متد عمل sell را برای بازیکن مورد نظر انجام میدهد و State جدید را بر اساس عمل انجام شده برمیگرداند.

```
def GetSellState(self,player:str):
    agent1=copy.deepcopy(self.Agent1)
    agent2=copy.deepcopy(self.Agent2)
    properties=copy.deepcopy(Properties)
    if player=="max":
        agent2.cash =agent2.cash + (properties[agent2.Position].Price *.9)
        Properties[agent2.Position].Owner==None
    elif player=="human":
        agent1.cash =agent1.cash + (properties[agent1.Position].Price *.9)
        properties[agent1.Position].Owner==None
    return State(agent1,agent2,properties)
```

- GetRentState() : : با توجه به بازیکن و حالت فعلی بازی، این متد عمل Rent را برای بازیکن مورد نظر انجام میدهد و State جدید را بر اساس عمل انجام شده برمیگرداند.

```
def GetRentState(self,player:str):
    agent1=copy.deepcopy(self.Agent1)
    agent2=copy.deepcopy(self.Agent2)
    if p (variable) agent2: Agent
                                    self.Agent1.Position].RentPrice
        agent2.cash +=Properties[self.Agent1.Position].RentPrice
    elif player=="max":
        agent2.cash -= Properties[self.Agent2.Position].RentPrice
        agent1.cash +=Properties[self.Agent2.Position].RentPrice

    return State(agent1,agent2,copy.deepcopy(Properties))
```

- GetDoNothingState() : : با توجه به بازیکن و حالت فعلی بازی، این متد عمل DoNothing را برای بازیکن مورد نظر انجام میدهد State جدید را بر اساس عمل انجام شده برمیگرداند. در واقع این متد کاری انجام نمیدهد و همان State را برمیگرداند.

```
def GetDoNothingState(self):
    agent1=copy.deepcopy(self.Agent1)
    agent2=copy.deepcopy(self.Agent2)
    newproperties=copy.deepcopy(self.Properties)
    return State(agent1,agent2,newproperties)
```

- GetChanceState(self,chancetype,roll) : وظیفه این متد ،تغییر Position هر بازیکن بر اساس مقدار تاس انداخته شده (roll) است که بعد از تغییر position ، State جدید برگشت داده میشود. اگر chancetype=”chancemin” باشد، انگاه position بازیکن از جنس انسان تغییر میکند ولی اگر chancetype=”chancemax” باشد انگاه position ربات تغییر میکند.

```
def GetChanceState(self,chancetype,roll:int):
    agent1=copy.deepcopy(self.Agent1)
    agent2=copy.deepcopy(self.Agent2)
    newproperties=copy.deepcopy(self.Properties)
    if chancetype=="chancemax":
        agent2.Position =(agent2.Position + roll)%len(Properties)
    elif chancetype=="chancemin":
        agent1.Position =(agent1.Position + roll)%len(Properties)

    return State(agent1,agent2,newproperties)
```

- GetSuccessors(self,statetype) : این متد بر اساس Statetype ،لیست عملیات ممکن برای هر State را برمیگرداند.
  همچنین statetype میتواند مقدار "human" را هم بگیرد که در این صورت عملیات های ممکن برای بازیکن از جنس انسان را برمیگرداند.

```
def get_successors(self,StateType):
    successors:list[State]=[]
    if StateType=="chancemin":
        for i in range(1,7):
            successors.append([self.GetChanceState("chancemin",i),"roll{0}".format(i)])
    elif StateType=="chancemax":
        for i in range(1,7):
            successors.append([self.GetChanceState("chancemax",i),"roll{0}".format(i)])
    elif StateType=="max":
        currentproperty=Properties[self.Agent2.Position]
        if self.Agent2.cash >= currentproperty.Price  and currentproperty.Owner==None and currentproperty.HasTax==False:
            successors.append([self.GetBuyState("max"),"buy"])
            successors.append([self.GetDoNothingState(),"DoNothing"])
        elif currentproperty.Owner==self.Agent1.Id:
            successors.append([self.GetRentState("max"),"rent"])
        elif currentproperty.Owner==self.Agent2.Id:
            successors.append([self.GetSellState("max"),"sell"])
            successors.append([self.GetDoNothingState(),"DoNothing"])
    elif StateType=="human":
        currentproperty=Properties[self.Agent1.Position]
        if self.Agent1.cash >= currentproperty.Price  and currentproperty.Owner==None and currentproperty.HasTax==False:
            successors.append([self.GetBuyState("human"),"buy"])
            successors.append([self.GetDoNothingState(),"DoNothing"])
        if currentproperty.Owner==self.Agent2.Id:
            successors.append([self.GetRentState("human"),"rent"])
        if currentproperty.Owner==self.Agent1.Id:
            successors.append([self.GetSellState("human"),"sell"])
            successors.append([self.GetDoNothingState(),"DoNothing"])
    return successors
```

Game : این کلاس محیط بازی را نشان میدهد و تصمیم گیری های ربات و انسان و وضعیت بازی را کنترل میکند .

```
class Game:
    def __init__(self,agents:list[Agent],currentState:State):
        self.Agents=agents
        self.CurrentState:State=currentState
```

- Agents : این پراپرتی لیست بازیکن ها را در خود نگه میدارد که برای این بازی دو نفره، عضو صفر لیست عامل انسان و عضو اول این لیست نیز بازیکن ربات میباشد.
- CurrentState : این پراپرتی State فعلی بازی را نگه میدارد.

- Expectiminimax(self,State,Depth,currentPlayer) : این متد مهم ترین متد بازی میباشد که با استفاده از هوش مصنوعی و الگوریتم جستجوی expectiminimax و با ارزیابی وضعیت فعلی بازی بهترین عمل را با توجه به تابع ارزیابی داده شده برای عامل هوشمند (ربات) انتخاب میکند. این متد به صورت بازگشتی درخت بازی را تولید میکند و از روی ان BestUtitily را پیدا میکند. جزئیات این الگوریتم و نحوه پیاده سازی ان در ادامه شرح داده خواهد شد.

```python
def ExpectiMiniMax(self,state:State,Depth,currentplayer:str):
    bestmove=None
    if Depth==0:
        return bestmove,self.EvalFunc(state)
    if currentplayer=="max":
        max_value = float('-inf')
        for successor in state.get_successors("max"):
            bestmove,value = self.ExpectiMiniMax(successor[0], Depth - 1,"chancemin")
            if value > max_value:
                bestmove=successor
                max_value=value
        return bestmove,max_value
    elif currentplayer=="chancemax":
        ExpectedValue = 0
        for successor in state.get_successors("chancemax"):
            bestmove,value =self.ExpectiMiniMax(successor[0], Depth - 1,"max")
            ExpectedValue =ExpectedValue + value* (1/6)
        return bestmove,ExpectedValue

    elif currentplayer=="chancemin":
        ExpectedValue = 0
        for successor in st
            bestmove,value                                                    ,"min")
            ExpectedValue =ExpectedValue + value* (1/6)
        return bestmove,ExpectedValue
    else:
        min_value = float('inf')
        for successor in state.get_successors("min"):
            bestmove,value = self.ExpectiMiniMax(successor[0], Depth - 1,"chancemax")
            if value < min_value:
                bestmove=successor
                min_value=value
        return bestmove,min_value
```

(variable) ExpectedValue: Any | Literal[0]

- MakeHumanDecision(self) : این متد وظیفه هندل کردن تصمیم گیری کاربر انسان را برعهده دارد. در واقع با استفاده currentstate بازی ، عملیات های ممکن برای انسان را پیدا میکند و کاربر با یکی ار عمل ها را انتخاب کند که بعد از انجام عمل ، currentstate بازی تغییر میکند.

```
def MakeHumanDecision(self):
    print("Current Position : ",Properties[self.CurrentState.Agent1.Position].Name)
    print("price : ",Properties[self.CurrentState.Agent1.Position].Price)
    if self.CurrentState.Properties[self.CurrentState.Agent1.Position].Owner != None:
        print("Owner : ",self.CurrentState.Properties[self.CurrentState.Agent1.Position].Owner)
    else:
        print("!has no owner")
    print("possible Actions: ")
    possibleactions=self.CurrentState.get_successors("human")
    for i in range(len(possibleactions)):
        print(i,")",possibleactions[i][1])
    selectedaction=int(input("Enter Your Action:"))
    if selectedaction < len(possibleactions) and selectedaction>=0 :
            print("you choosed action: ",possibleactions[selectedaction][1])
            input("press any key to continue...")
            self.CurrentState=copy.deepcopy(possibleactions[selectedaction][0])
            print("your cash(after apply action): ",self.CurrentState.Agent1.cash)
            input("press any key to continue")
```

- MakeRobotDecision(self) : این متد وظیفه هندل کردن تصمیم گیری ربات را برعهده دارد. ابتدا بر اساس وضعیت فعلی بازی ،

متد Expectiminimax(currentState,depth,”max”) صدا زده میشود و بهترین عمل بر اساس ان انتخاب میشود و currentState بازی تغییر میکند.

```
def MakeRobotDecision(self):
    bestaction,bestvalue=self.ExpectiMiniMax(self.CurrentState,2,"max")
    print("Current position : ",Properties[self.CurrentState.Agent2.Position].Name)
    print("robot1 choosed action:",bestaction[1])
    input("press any key to continue...")
    self.CurrentState=bestaction[0]
```

- RollDice(self) : این متد وظیفه انداختن تاس را برعهده دارد . در واقع با استفاده از پکیج Random در پایتون ، یک عدد رندوم بین 1تا 6 برگشت داده میشود.

```
def RollDice(self):
        return randint(1, 6)
```

- Play(self) : در این متد بازی شروع میشود و بازیکنان نوبتی (با شروع از بازیکن انسان) تاس می اندازند و بر اساس تاس انداخته شده، position بازیکن

تغییرمیکند و متد MakeHumanDecision() برای بازیکن انسان و متد MakeRobotDecision() برای ربات صدا زده میشود تا تصمیم گیری کنند .

```
def Play(self):
    i=0    #current playerId
    while(True):
        roll=self.RollDice()
        print(self.Agents[i].Name , "rolled " ,roll,"\n")
        input("press any key to continue...")
        self.Agents[i].Position =(self.Agents[i].Position + roll) % len(Properties)
        if self.Agents[i].Type=="human":
            self.CurrentState.Agent1.Position= (self.CurrentState.Agent1.Position + roll)% len(self.CurrentState.Properties)
            self.MakeHumanDecision()
        elif self.Agents[i].Type=="robot":
            self.CurrentState.Agent2.Position= (self.CurrentState.Agent2.Position + roll)% len(self.CurrentState.Properties)
            self.MakeRobotDecision()
        if self.CurrentState.IsTerminal():
            print("Game Over")
            if self.CurrentState.Agent1.cash>0 :
                print(self.CurrentState.Agent1.Name ," won!!!!!")
            else:
                print(self.CurrentState.Agent2.Name," won!!!!!!")
        i=(i+1) % len(self.Agents)
```

الگوریتم Expectiminimax برای بازی مونوپولی:

- بازیکن ربات به عنوان بازیکن max محسوب میشود و طبق این الگوریتم باید utility ربات ماکسیسمم شود.
- سه نوع state داریم : max , min ,chance . بازیکن حریف ،min محسوب میشود و در chance هم فقط تاس انداخته میشود.
- حالت chance هم خود دو نوع است : chancemin که برای پرتاب تاس بازیکن min و chancemax هم برای پرتاب تاس بازیکن max میباشد .
- اگر "currentplayer="max باشد انگاه value ان از ماکسیمم گرفتن state های فرزند ان بدست می آید.
- اگر "currentplayer="min باشد انگاه value ان از مینیمم گرفتن state های فرزند ان بدست می آید.
- اگر " currentplayer="chancemin یا "currentplayer="chancemax باشد انگاه value ان از امیدریاضی state های فرزند ان بدست می آید.
- در پایان نیز ،بهترین اکشن برگشت داده میشود.

تابع ارزیابی:

```
def EvalFunc(self,state:State):
    sumValProps=0
    for property in state.Agent2.Properties:
        sumValProps +=property.Price
    value=state.Agent2.cash + sumValProps
    return value
```

- طبق این تابع ، ارزش هر state برای بازیکن max برابر است با :

ارزش = مجموع ارزش دارایی ها + مقدار پول نقد

شروع بازی:

```
def ConfigGame():
    #Config Agents :
    player1Name=input("Enter Player name:")
    agent1=Agent("human",0,player1Name,0,1000)
    agent2=Agent("robot",1,"robot1",0,1000)
    firststate=State(agent1,agent2,Properties)
    #Define Game:
    game=Game([agent1,agent2],firststate)
    return game

def Main():
    game=ConfigGame()
    game.Play()


Main()
```

منابع

- صالح شکور(9913016) برای محاسبه تابع ارزیابی
- امیر شهبازی(9812033) برای الگوریتم expectiminimax
- ChatGpt برای الگوریتم expectiminimax