Inteligencia artificial projeto 2: multiagente

Guilherme Costa Pateiro GRR20197152

parte 1: função de avaliação

```
"*** YOUR CODE HERE ***"

comida = newFood.asList()
distMinimaComida = 99999999
for c in comida:
    distancia = util.manhattanDistance(newPos, c)
    if distancia < distMinimaComida:
        distMinimaComida = distancia

distMinimaFantasma = 99999999
for c in newGhostStates:
    distancia = util.manhattanDistance(newPos, c.getPosition())
    if distancia <= 1:
       return -99999999
    if distancia < distMinimaFantasma:
        distMinimaFantasma = distancia

return successorGameState.getScore() + 1/(float(distMinimaComida)+1)  - 1/(float(distMinimaFantasma)+1)
```

A função se baseia em pegar a distancia mínima entre o pacman gerando a “distancia mínima comida”.

Depois pegar a distancia mínima dos fantasmas, caso dentro da recursão o pacman acabe colidindo com um fantasmas, dentro da recursão a função devolvera -99999999 ou seja o pior resultado possível, caso não haja colisão ele devolver a ”distancia mínima fantasma”.

A função de avaliação se baseia em pegar a pontuação do tabuleiro, caso ele coma uma comida ou fantasma a pontuação aumenta então essa função beneficia o pacman por fazer tais coisas, de pega + 1/ distancia mínima da comida (o + indica que é bom diminuir a distancia+1) e - 1/a distancia mínima dos fantasmas+1 (o – indica que é ruim diminuir a distancia).

O +1 é por que ha risco de divisão por zero, o +1 evita isso.

A função leva em consideração o inverso por que quando menor o numero maior sera o impacto da função.

parte 2: minimax

```
def minimax(agente, profundidade, gameState):
    listaEstados = []
    if gameState.isLose() or gameState.isWin() or profundidade == self.depth:
        return "stop" , self.evaluationFunction(gameState)

    if agente == 0:
        for estado in gameState.getLegalActions(agente):
            _ , custo = minimax(1, profundidade , gameState.generateSuccessor(agente, estado))
            proximoEstado = (estado, custo)
            listaEstados.append(proximoEstado)
        return max(listaEstados ,key=ret_2nd_ele)


    else:
        proximoAgente = agente + 1
        if gameState.getNumAgents() == proximoAgente:
            proximoAgente = 0
        if proximoAgente == 0:
            profundidade += 1
        for estado in gameState.getLegalActions(agente):
            _ , custo = minimax(proximoAgente, profundidade , gameState.generateSuccessor(agente, estado))
            proximoEstado = (estado , custo)
            listaEstados.append(proximoEstado)
        return min(listaEstados, key=ret_2nd_ele)


agentePac = 0
acao, valor = minimax(agentePac, 0, gameState)
return acao

util.raiseNotDefined()
```

Primeiro a função cria a lista de estados, uma lista auxiliar que armazena todas as tuplas de (ação, custo) sendo ação a direção andada naquele momento, ex “left”, e o custo é o retorno das chamadas recursivas após chegar na self.evaluationFunction

A ação é útil pois indica qual a direção que deve ser andada por pacman na saída da função minimax, o custo serve para escolher qual melhor ação a ser tomada
ação só é útil na saída de minimax por isso dentro da recursão “ação” é descartada

A lista armazena os custos de saída da recursão e é muito útil pois o elemento mínimo pode ser pego utilizando as funções min e max do python

Minimax se baseia em recursivamente analisar a menor perda possível, para isso quando o algoritmo age sobre pacman ele maximiza o resultado e quando ele passa para os fantasmas eles minimizam

Importante apontar que a profundidade só aumenta quando todas as entidades são analisadas nessa camada

parte 3 :alpha-beta

```
if gameState.isLose() or gameState.isWin() or profundidade == self.depth:
    return "stop" , self.evaluationFunction(gameState)

if agente == 0:
    v = float("-inf")
    for estado in gameState.getLegalActions(agente):
        _ , custo = alphabeta(1, profundidade , gameState.generateSuccessor(agente, estado),a,b)
        if custo > v:
            v = custo
            melhorEstado = estado
        if v > b:
            return estado , v
        a = max(a, v)
    return melhorEstado, v

else:
    proximoAgente = agente + 1
    if gameState.getNumAgents() == proximoAgente:
        proximoAgente = 0
    if proximoAgente == 0:
        profundidade += 1

    v = float("inf")
    for estado in gameState.getLegalActions(agente):
        _ , custo = alphabeta(proximoAgente, profundidade , gameState.generateSuccessor(agente, estado), a , b)
        if custo < v:
            v = custo
            melhorEstado = estado
        if v < a:
            return estado , v
        b = min(b, v)
    return melhorEstado, v
```

o esqueleto é parecido com o minimax porem ao invés de inserir os elementos em uma lista, é feito a logica de comparação da alpha-beta, são feitas algumas comparações extras para indicar qual o melhor estado encontrado e caso V nunca supere alpha (ou beta dependendo da entidade) o algoritmo devolverá o melhor estado encontrado (no caso do pacman o maior e no caso dos fantasmas o pior)

parte 4: expectimax

```
"*** YOUR CODE HERE ***"
def ret_2nd_ele(tuple_1):
    return tuple_1[1]

def expectimax(agente, profundidade, gameState):
    listaEstados = []
    if gameState.isLose() or gameState.isWin() or profundidade == self.depth:
        return "stop" , self.evaluationFunction(gameState)

    if agente == 0:
        for estado in gameState.getLegalActions(agente):
            _ , custo = expectimax(1, profundidade , gameState.generateSuccessor(agente, estado))
            proximoEstado = (estado, custo)
            listaEstados.append(proximoEstado)
        return max(listaEstados ,key=ret_2nd_ele)


    else:
        proximoAgente = agente + 1
        if gameState.getNumAgents() == proximoAgente:
            proximoAgente = 0
        if proximoAgente == 0:
            profundidade += 1

        custototal = 0
        for estado in gameState.getLegalActions(agente):
            _ , custo = expectimax(proximoAgente, profundidade , gameState.generateSuccessor(agente, estado))
            custototal += custo
        return "stop", custototal / len(gameState.getLegalActions(agente))
```

O esqueleto é parecido com a minimax, a lista de estados voltou (para o pacman).

A parte do pacman é idêntica ao minimax, a parte inédita é a parte dos fantasmas, ao invés de salvar os custos em uma lista e devolver o resultado perfeito , ele tira a media de todos os estados encontrados. Como a “ação” é descartada no retorno dos fantasmas, os fantasmas devolverão sempre a ação “stop” sem que haja uma perda no algoritmo.

Parte 4: função de avaliação melhor

```
"*** YOUR CODE HERE ***"
posicaoPac = currentGameState.getPacmanPosition()
aux = currentGameState.getFood()
comida = aux.asList()


distMinimaComida = 99999999
for c in comida:
    distancia = util.manhattanDistance(posicaoPac, c)
    if distancia < distMinimaComida:
        distMinimaComida = distancia

distMinimaFantasma = 99999999
for c in currentGameState.getGhostStates():
    distancia = util.manhattanDistance(posicaoPac, c.getPosition())
    if distancia <= 1:
        return -99999999
    if distancia < distMinimaFantasma:
        distMinimaFantasma = distancia

bolao = currentGameState.getCapsules()
boloes = len(bolao)

return 100 * currentGameState.getScore() + 1/(float(distMinimaComida)+1)  - 1/(float(distMinimaFantasma)+1) - boloes
```

A função de avaliação segue a mesma logica da primeira com algumas modificações:

- ela leva mais em consideração a pontuação do jogador, pegar uma comida aumenta consideravelmente a pontuação, não pegar comida diminui a pontuação
- agora ela considera os bolões. Comer os bolões é incentivado com o decremento da variavel bolões, e após isso o pacman pode ganhar pontuação ao matar o fantasma, algo que é incentivado com o sistema de pontuação

Ainda ha a mesma logica de distancia comida e distancia fantasma

Estados que levam a derrota de pacman ainda tomam como retorno -99999999 indicando que é a pior opção possível