Pacman, captura la bandera

Autors: - Pol Sala Lidón - Amat Martínez Vilà

Visió general (estratègia)

Després de ralitzar diverses proves i disenys, hem optat per desenvolupar dos agents per separat amb comportaments diferents (**ofensiu** i **defensiu**) però amb una part genèrica la qual podràn fer servir qualsevol dels dos a través d'herencia.

A l'hora de seleccionar quin moviment ha de realitzar un agent determinat ho farà a través d'un algoritme **Expectimax** assumint la posició més aproximada de l'enemic en base a les creences que te les quals seran aproximacions calculades a través de les funcions **updateBeliefs i observe**.

```
def updateBeliefs(self, enemy):
    """
    Comprova totes les possibles posicions succesores i que sigui legal el
    moviment i reparteix de manera uniforme la distribucio de probabilitats
    """
    newBelief = util.Counter()
    # legalPositions es una llista de tuples (x,y)
    for oldPos in self.legalPositions:
        # Obtenim la nova distribució de probabilitats
        newPosDist = util.Counter()

        # Mirem les possibles posicions succesores
        for i in [-1, 0, 1]:
            for j in [-1, 0, 1]:
                if not (abs(i) == 1 and abs(j) == 1):
                    pos_pos = (oldPos[0] + i, oldPos[1] + j)
                    if pos_pos in self.legalPositions:
                        newPosDist[pos_pos] = 1.0

        # Normalitzem les probabilitats
        newPosDist.normalize()

        # Noves creences
        for newPos, prob in newPosDist.items():
            # Update the probabilities for each of the positions.
            newBelief[newPos] += prob * self.beliefs[enemy][oldPos]

    # Normalize and update the belief.
    newBelief.normalize()
    self.beliefs[enemy] = newBelief
```

En quan els comportaments específics, l'agent ofensiu l'hem disanyat a través d'una funció d'evaluació perque el seu comportament es basi en entrar i moures per el camp enemic evitant ser atrapat, per alte banda el defensiu ha estat disenyat per mantenirse en el camp defensant i perseguint qualsevol enemic que entri al territori.

Aquest diseny també permet que en cas de nessesitat els dos agents puguin executar el mateix rol, per exemple, atacar els dos quan no hi hagi enemics al territori i tinguem el **power-up** actiu.

Acotament de les cerques

Degut a que a l'hora d'obtenir les distancies dels sons ens retornava masses posicions a evaluar i que algunes d'elles eren impossibles hem decidit acotarles amb les seguents regles:

- Si una lectura es troba en una posició en la que hauria d'estar l'enemic en forma de pacman la descartem (**prob=0.**) si no ho es, tot això verificant l'estat a través de la funció:

`python gameState.getAgentState().isPacman` - Comprovar si una lectura de so es troba dins el nostre rang de visió $< \mathbf{6}$, en aquest cas ens hauria d'apareixer dins les lectures reals, si no ho fa la podem descartar ($\mathbf{prob = 0.}$).

```
def observe(self, enemy, observation, gameState):
    """
    Funció d'acotament per determinar més exactament la possible
    posició enemiga (creences).
    @param: gameState state of the game.
    @param: observation int list of 4 elements with noisy distance
            between current agent and all agents.
    """
    # Obtenim la distància al enemic
    noisyDistance = observation[enemy]

    # La nostra posició
    myPos = gameState.getAgentPosition(self.index)

    # Diccionari per a guardar les creences per al enemic actual
    newBelief = util.Counter()

    # Actualitzem les creences de les posicions legals del tauler.
    for pos in self.legalPositions:
        # Distancia real entre l'agent actual i la posicio iteració
        trueDistance = util.manhattanDistance(myPos, pos)

        # Probabilitat tenint en compte la distancia real i la probable
```

```
            emissionModel = gameState.getDistanceProb(trueDistance, noisyDistance)

            # Podem descartar que una posicio sigui real
            # comrovant el tipus d'agent que es pacman o fantasma i sapiguent
            # a quin camp es troba.
            pac = self.inOurTerritory(pos)

            # Si la distancia real es inferior a 6 la descartem perque tindria
            # visio de l'objectiu i no estaria al vector de distancies de sons
            # si no a les distancies reals
            if trueDistance <= 5:
                newBelief[pos] = 0.
            elif pac != gameState.getAgentState(enemy).isPacman:
                newBelief[pos] = 0.
            else:
                newBelief[pos] = self.beliefs[enemy][pos] * emissionModel

        # Si no tenim creences inicialitzem de manera uniforme per cada posicio
        # altrament normalitzem i actualitzem amb les noves creences
        if newBelief.totalCount() == 0:
            self.initializeBeliefs(enemy)
        else:
            newBelief.normalize()
            self.beliefs[enemy] = newBelief
```

Comportament dels agents

Comportament Ofensiu

```
def chooseAction(self, gameState):
    enoughFood = 7

    # Si tenim prou menjar és moment de retirar-nos a un lloc segur
    self.retreating = gameState.getAgentState(self.index).numCarrying > enoughFood
    return ParentAgent.chooseAction(self, gameState)
```

El comportament ofensiu es basa en entrar en el territori recollir tot el menjar possible i no ser "caçat", per això la nostra funció d'evaluació es basa en trobar un equilibri entre risc i recompensa on la nostre prioritat és no ser "caçats".

Un altre factor a tenir en compte era que en ser caçat dins el territori enemic, equivalia a perdre els punts recolectats per tant vam decidir que a cert punt l'agent retornés al seu territori per tal de conservar aquests punts a menys que tinguéssim el *power-up* actiu.

La reflexio principal d'aquest agent i de l'agent defensiu en mode atac es basa en prioritzar l'obtenció del *power-up* ja que facilita l'obtenció de menjar. Quan el nostre equip ja te una puntuació mitjanament alta generalment el menjar que queda esta en posicions més complicades d'accedir, aquest es el moment en que els dos agents poden pasar a l'atac ja que sera més facil que un pugui tornar amb menjar.

Comporament Defensiu

```
def chooseAction(self, gameState):
    # Mirem s'hi ha algun enemic en forma de pacman (atacant)
    invaders = [enemy_agent
                for enemy_agent in self.enemies
                if gameState.getAgentState(enemy_agent).isPacman]

    # Mirem si tenim el power-up actiu.
    powerTimes = [gameState.getAgentState(enemy).scaredTimer
                  for enemy in self.enemies]

    # Si no hi ha enemics atacant i tenim el power-up actiu pasem l'agent
    # a l'atac
    self.offensing = not invaders and min(powerTimes) > 8

    return ParentAgent.chooseAction(self, gameState)
```

El comportament defensiu es basa en un equilibri entre defensa i atac, es a dir, la seva funció principal és defensar el territori dels enemics que entrin però també podra atacar si es nesesari (quan no hi hagi gaire risc).

Per determinar si defensar o atacar ens basem en si hi ha algun enemic en forma de pacman, si és el cas en que l'enemic esta etacant per tant l'agent entrara en mode defensiu altrament entrara en mode atac si tenim el *power-up* actiu.

En cas d'estar en mode defensiu la funció d¡evaluacio buscara el moviment que minimitzi la distancia entre l'agent actual i l'enemic més proper, altrament en cas d'estar atacant, utilitzara la funcio de l'agent atacant i es mantindrà atacant fins que torni a tocar defensar.