Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name:
Podina Tudor
Tudor Roxana

Group:
30434

Email:
tpodina@gmail.com / Podina.Do.Tudor@student.utcluj.ro
roxi13.tudor@gmail.com / Tudor.Co.Roxana@student.utcluj.ro

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Table 1: Lab scheduling

Activity	**Deadline**
Searching agents, Linux, Latex, Python, Pacman	W_1
Uninformed search	W_2
Informed Search	W_3
Adversarial search	W_4
Propositional logic	W_5
First order logic	W_6
Inference in first order logic	W_7
Knowledge representation in first order logic	W_8
Classical planning	W_9
Contingent, conformant and probabilistic planning	W_{10}
Multi-agent planing	W_{11}
Modelling planning domains	W_{12}
Planning with event calculus	W_{14}

Lab organisation.

1. Laboratory work is 25% from the final grade.
2. There are three deliverables in total: 1. Search, 2. Logic, 3. Planning.
3. Before each deadline, you have to send your work (latex documentation/code) at moodle.cs.utcluj.ro
4. We use Linux and Latex
5. Plagiarism: Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more.

Chapter 1

A1: Search - Pacman Wants a Family

1.1 Preview

What was the goal? The goal of this assignment was to allow us to take a deep dive into the Pacman framework.
In doing so, we had the opportunity to understand how each method gets called and when. Moreover, we got to experience first-hand what happens under the hood of the framework.
Besides the framework-side of the assignment, we were also given the chance to experiment with multiagents and how they work, as well as research how NPC's work, not just within the Pacman game, but as a whole.
As a final mention, we also wanted to do something extra and had the chance to read some PhD papers. And while they proved too difficult to implement on short notice (some were even outside the subject of this semster), they proved to be very insightful.

1.2 Keyboard agent

1.2.1 Gameplay

We decided on switching the role of Pacman from being chased to chasing either Ms. Pacman or food.
This allowed to introduce Ms. Pacman as an agent that could be controlled via keyboard. We also focused on giving Ms. Pacman a hard time by allowing Pacman to eat her food, thus lowering her score.

1.2.2 Objective

Survive. On a serious note, what the player (Ms. Pacman) has to do is very similar to the default version of Pacman: eat food and avoid ghosts. There is an improvement with respect to the challenge, since Ms. Pacman also has to deal with Pacman eating her food, causing her to lose out on points.

1.3 Multiagents

1.3.1 Objective

As mentioned before, Pacman gets an upgrade within our version of the game, meaning that he is the "antagonist" now. We decided to replace the concept of ghosts with Pacmen that share a common goal of eating as much as possible and chasing Ms. Pacman.

1.3.2 How do they work?

At first, we needed to figure out how to compute the next steps based on the current location of Ms. Pacman. With this in mind, we explored the concept of *ghost agents* which have a *getDistribution* method. This method generates a distribution based on a list of the best actions that Pacman can take*.
We also focused on having two types of agents. One of them works using the *manhattan distance* while the other uses the *maze distance* towards a goal. It is also worth noting that we tweaked the goal to be composed of food and Ms. Pacman. This means that Pacman either chases food or Ms. Pacman depending on who is closer. Also, we played a little with the idea of attaching a probability of being chosen as a goal for each of the two targets. While our implementation allows this, we did not have enough time to implement and test this properly, so we will leave it for a future improvement.
Note*: There is something in the framework that we decided not to change, but we are aware of it: the action that Pacman eventually takes is randomly picked.

Chapter 2

A2. Planning

2.1 How we chose the task

Initially, we wanted to combine the idea of having Ms. Pacman with random maze generation and junior Pacmen. Instead of overloading our assignment with ambitious ideas that we may or may not have implemented properly, we decided to stick with adding Ms. Pacman as the agent controlled by the player and multiple Pacmen replacing the default ghosts.
After some brainstorming, we also decided on having Pacman eat Ms. Pacman's food.

2.2 Sub-task distribution

At first, we did some pair programming, mostly to setup the project, do the necessary research and explore the framework to decide upon what needs to be done and what is already implemented.
After that, we tried to parallelize the development process and distributed the tasks as such:

1. Random maze generation - discarded - Podina. T
2. Ms. Pacman and Pacmen animations - Tudor. R
3. Multiagents decision making - Podina. T
4. Multiagents food eating - Tudor. R
5. User experience improvements - Podina. T, Tudor R.
6. Documentation:
   (a) Writing and structuring - Podina. T
   (b) Quality control and refactoring - Tudor. R

Bibliography

1. `http://ai.berkeley.edu/multiagent.html` - base project
2. `https://github.com/lb5160482/Pacman-Search` - useful for examples and PositionSearchProblem which we used for the Pacmen agents

Appendix A

Your original code

Don't be a cheater! Cheating affects your colleagues, scholarships and a lot more. This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained.

```
class MazeDirectionalPacman( PacmanAgent ):
    "A Pacman agent that prefers to eat food or chase Ms. Pacman."
    def __init__( self, index, prob_food=0.5, prob_msPacman=0.5 ):
        self.index = index
        self.prob_food = prob_food
        self.prob_msPacman = prob_msPacman

    def getDistribution( self, state ):
        # Read variables from state
        legalActions = state.getLegalActions( self.index )
        pos = state.getGhostPosition( self.index )

        speed = 1

        actionVectors = [Actions.directionToVector( a, speed ) for a in
    legalActions]
        newPositions = [( pos[0]+a[0], pos[1]+a[1] ) for a in actionVectors]
        msPacmanPosition = state.getPacmanPosition()
        foodGrid = state.getFood()

        # Select best actions given the state
        distancesToMsPacman = [mazeDistance( pos, msPacmanPosition, state )
    for pos in newPositions]
        distancesToFood = []
        for y in range(foodGrid.height):
            for x in range(foodGrid.width):
                if foodGrid[x][y] == True:
                    distancesToFood.extend( [mazeDistance( pos, (x, y),
    state )] for pos in newPositions )

        msPacmanMinDistance = min( distancesToMsPacman )
        if distancesToFood:
            foodMinDistance = min( distancesToFood )
            bestScore = min( msPacmanMinDistance, foodMinDistance  )
            bestProb = self.prob_food if foodMinDistance <
    msPacmanMinDistance else self.prob_msPacman
        else:
            bestScore = msPacmanMinDistance
            bestProb = self.prob_msPacman

```

```
        bestActions = [action for action, distance in zip( legalActions,
    distancesToMsPacman ) if distance == bestScore]
        bestActions.extend( [action for action, distance in zip(
    legalActions, distancesToFood ) if distance == bestScore] )

        # Construct distribution
        dist = util.Counter()
        for a in bestActions: dist[a] = bestProb / len(bestActions)
        for a in legalActions: dist[a] += ( 1-bestProb ) / len(legalActions)
        dist.normalize()
        return dist
```

Note 1: DirectionalPacman works the same, only it uses the manhattan istance, rather than the maze distance.
Note 2: The mazeDistance() method is defined in the repository mentioned here 2. We used helpers from the said repository and made a slight adjustment to the method to work with our code. Here is the implementation from our code.

```
def mazeDistance(point1, point2, game_state):
    prob = search.PositionSearchProblem(game_state, start=point1, goal=
    point2, warn=False, visualize=False)
    return len(search.ids(prob))
```

Note 3: as a future improvement to this method, it would be interesting to allow the user to set which search algorithm to use for the mazeDistance() method. By default, we are using the Iterative Deepening Search. Maybe the search algorithm would be swapped for a better one as the user sets a higher difficutly to the game.