PacMan v PacMan

Pete and Xiaodi

October 13, 2010

1 Game rules

In CS 154, you will design the AI for a competitive pac-man game. Different from the original pac-man, there are no ghosts. Instead, you will compete against pac-men from other teams. The objective of the game is simple: eat the most pac-dots.

The game will have two iterations; in both iterations, you will be given the same information at each time step t of the game: a 2D array $\mathbf{m}_t$, the state of the pac-man of team $k \in \{0, 1\}$, $\mathbf{s}_t(k)$; and the state of the fruit $\mathbf{f}_t$.

The value of each tile $m_t(x, y)$ at x, y location indicates the following:

- the content of tile (x, y): it could be either a wall, a tunnel, empty tile, or a tile that contains one pac-dot.
- contains a tunnel to the other side of the maze

The vector $\mathbf{s}_t(k)$—where $k = 0$ is your pacman and $k = 1$ is the opponent pacman—include the following information:

- the pixel location of pac-man k
- the pixel moving speed of pac-man k
- whether or not pac-man k is a power pac-man
- whether or not pac-man k is currently disabled

The vector $\mathbf{f}_t$ include the following information:

- the pixel location of the fruit.

Given $\mathbf{m}_t, s_t(0), s_t(1)$, you will return one of the following 4 commands at time step t to the game: $c_t = \{u, d, l, r, h\}$, meaning **up, down, left, right and hold (stop)**, respectively. For example, in Fig.1, a sequence of actions starting at time t is given by $c_t = d, c_{t+1} = d, c_{t+2} = d, c_{t+3} = d$, which, for the deterministic motion case, results in pac-man moving from Fig.1.A to Fig.1.B. Once c_t is returned to the game, the game will then return $\mathbf{m}_{t+1}, s_{t+1}(0), s_{t+1}(1)$.

The two iterations (first iteration for milestone, and second for the final) of the game are as follows:

First iteration : you will be asked to design an AI which finds an optimal path through the maze. However, you will be competing simultaneously with another pac-man, which will also try to eat the most dots. Of course, your algorithm might need to adjust strategy based on what the other pac-man is currently doing. Specifically, each dot worths 10 points. In a rare case where two pac-men reach the same tile at the same time, each one receives 5 points for the dot.

Second iteration : A fruit will occasionally bump into the maze from one tunnel and exit from the other. The fruit does not contain any point reward, however, it turns a pac-man into a power-pac-man, which can eat (disable) the other pac-man for a set amount of time. During this time, the able-bodied power-pac-man can continue to eat pac-dots, and increase his lead. In this stage, you will also need to consider motion noise which means that the result of the command $c(t)$ is stochastic; for instance, the input $c(t)$ might not affect $s_l(t)$ in the intended fashion; your pac-man might move in a different direction than commanded, or might not move at all. Finally, all of these new features will be tested on a new testing maze (not available until the day of tournament).

The game will end once all of the pac-dots are eaten, or if the AIs are especially bad, after a set amount of time has elapsed.

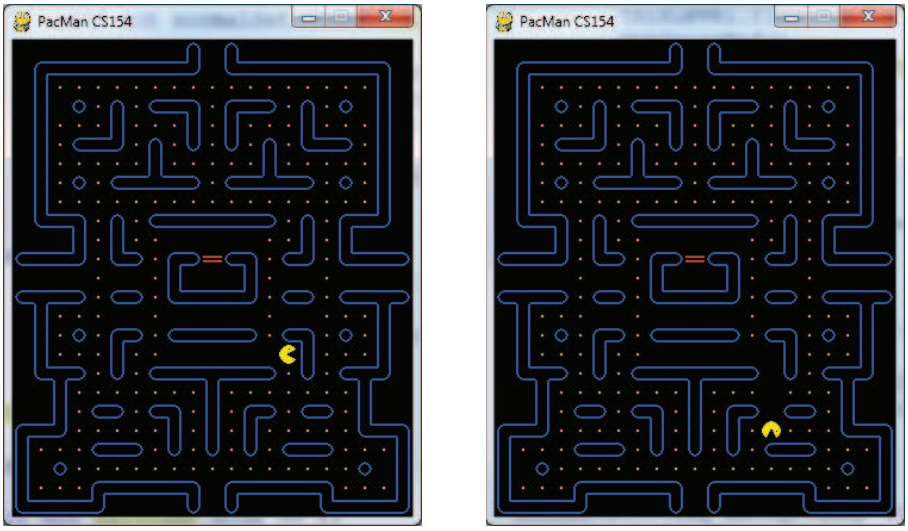

Figure 1: fdsa

2 Data structure

You will write your own `team1AI` class for your pac-man. You will have the chance to initialize your AI in the beginning of the game:

```
team1 = team1AI(thisLevel.map, curPos, rivalPos)
```

`thisLevel.map` is the map of the world. `curPos` is the position of your tile; `rivalPos` is the tile of your rival.

We do not limit the initialization time. But it should be reasonable in terms of complexity and memory usage (e.g. it should take less than 1 min running on a personal computer).

In every round, your pac-man calls `think` function of your AI, which is defined as following:

```
def think(self, curMap, curPos, rivalPos, curGraceTime):
```

`curMap` is $\mathbf{m}_t$, the 2D array of maze. $m_t(x, y)$ indicates the content within tile at (x, y). One tile can have one of the following values: a normal pellet - 2; horizontal door - 20; vertical door - 21; any $100 \leq m_t(x, y) \leq 199$ is a wall tile.

`curPos` is the position of your tile; `rivalPos` is the tile of your rival. `curGraceTime` shows the total grace time you have used. If it exceeds 5000 ms, you will lose the game. The return value of your function should be one character from the set $\{u, d, l, r, h\}$.

For more details, please see `Move` function of the `pacman` class

3 Language requirements

This project is based on Python $2.6.6$ and Pygame $1.9.1$. We will provide most of the sample code for the game. You need to implement your own `findPath` function that controls a PacMan.

If you're new to Python, we recommend a Linux or Windows based environment. The installation of Pygame on a Mac can be tricky.

Python 2.6.6 can be downloaded here: `http://www.python.org/download/releases/2.6.6/`. Pygame 1.9.1 provides the necessary classes for developing this game. It is also freely available at their website: `http://pygame.org/download.shtml`. For a windows based machine, you need the following files (choose the 32-bit version even if you're running 64-bit Windows):

- Windows x86 MSI Installer (2.6.6)
- pygame-1.9.1.win32-py2.6.msi

Personally I like the Eclipse + PyDev environment. It provides excellent customization, code management, and debug tools.

For ubuntu linux, python comes preinstalled. One merely needs to execute the command sudo apt-get install python-pygame, and then download the pac-man game which we will provide.