Module II

Programming with Pacman

Game Programming and Robotics - Fall' 2013
IBA, Karachi

Ms. Pacman Competition

- Ms Pac-Man, the classical arcade game from the 80's is one of the best-known video games ever made. In the Ms Pac-Man vs Ghosts competition you are asked to create computer programs (controllers) to play the game as either Ms Pac-Man or the ghosts. Your controller, which will have to provide a direction at every game tick (40 milliseconds), will then compete against all the controllers submitted by other competitors to determine the most effective strategy.

About Pacman

- The game consists of four mazes (A, B, C and D), which are played in order; when maze D is cleared (i.e., all pills have been eaten by Ms Pac-Man), the game continues with maze A and so on until the game is over.
- The goal of Ms Pac-Man is to obtain the highest possible score by eating all the pills and power pills in the maze (and thus advancing to the next stage).
- Each pill eaten scores 10 points, each power pill is worth 50 points.
- Ms Pac-Man's quest is opposed by the four ghosts: Blinky (red), Pinky (pink), Inky (green) and Sue (brown)
- At the start of each level, the ghosts start in the lair in the middle of the maze and, after some idle time, enter the maze in their pursuit of Ms Pac-Man. Their goal is to eat Ms Pac-Man and each time this happens, a life is lost.
- There are four power pills in each of the four mazes, which, when eaten, reverse the direction of the ghosts and turn them blue; they may now be eaten for extra points. The score for eating each ghost in succession immediately after a power pill has been consumed starts at 200 points and doubles each time, for a total of 200+400+800+1600=3000 additional points.
- The ghosts' behavior patterns are different and include semi-random movement, preventing the use of patterns to beat each level.
- The more advanced the level, the shorter the ghost edible times become, making the levels progressively more difficult. When the edible period runs out, the ghosts start flashing blue and white. The player (or agent) needs to be careful at this stage to avoid losing lives.
- When all the pills and power pills have been cleared, the game moves on to the next maze.

Agent

- What is an ‘Agent’?
  - An entity that can
    - Sense
    - Think
    - Act

Type of agents in Pacman

- Two type of controllers can be developed:
- **Pacman Controller:**
  - The goal of a Ms Pac-Man controller is to maximize the score of the game. In the competition, it is the average score against multiple ghost teams that counts and the winning controller is the one which obtains the highest total average score.
- **Ghost-team Controller:**
  - The goal of a ghost-team controller is to minimize the score obtained against it by the Ms Pac-Man controllers. The winning ghost team will be team with the lowest average score against it.

How can a pacman controller be intelligent?

Game Environments

- Discrete vs Continuous
- Deterministic vs Stochastic
- Fully Observable vs Partially observable
- Adversarial vs Non-adversarial

Pacman Environment

- So, pacman is:
  - Discrete or Continuous?
  - Fully observable or partially observable?
  - Deterministic or Stochastic?
  - Adversarial or Non-adversarial?

Pacman Environment

- So, pacman is:
  - Discrete or Continuous?
    - Discrete
  - Fully observable or partially observable?
    - Fully Observable
  - Deterministic or Stochastic?
    - Deterministic
  - Adversarial or Non-adversarial?
    - Adversarial

Game Environments

- What about Chess?
- What about Sumo Wrestling?
- And…..RoboCup?

Building a Pacman Controller

Starting with Pacman Simulator

1. Download

- Download the latest version of the software from http://www.pacman-vs-ghosts.net/software and unzip it in a convenient location on your computer.

2. Create a Project

- Open Eclipse and create a new (empty) Java project.

3. Import files

- Click on File, then Import ... and choose General, then File System.
- In the 'From directory' dialog, choose the folder where you unzipped the software.
- In the 'Into folder' dialog, choose the project you just created.
- Click Finished.

4. Run the Code

- You can execute the code by running the class *Executor.java*. This class contains a main method with numerous options for execution. Just uncomment the one you wish to run.

Running sample controllers

- Open Executor.java
- Make sure the following line is un-commented while the rest are commented:

```
exec.runGameTimed
  (new StarterPacMan(),new StarterGhosts(),visual);
```

- Run Executor.java

Running sample controllers

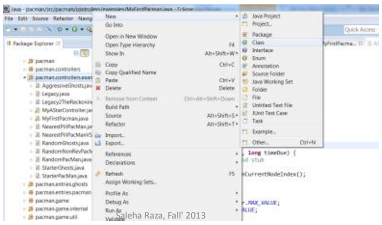
- Try running *HumanController* now !

Executor

- The Executor runs a game by instantiating two controllers (one for Ms. Pac-Man, one for the Ghosts) and creating a new game object. It then supplies each controller with a copy of the game object and then waits for 40ms for each controllers to compute their action.
- After the time is up, the actions computed by the controllers are used to advance the game state.
- This is continued until the end of the game (or the limit has been reached).

Creating a new Controller

- Open pacman workspace in Eclipse
- Add a new class named ‘MyFirstPacman’ in pacman.controllers.examples



Extending abstract ‘Controller’ class

- Each controller extends the **abstract class** Controller. The only method that needs implementation is
  - getMove(Game game, long timeDue)
- The game object is a copy of the current game state and timeDue signals the point in time the game advances; a controller should return an action before the time is up.

Returning pacman moves

- The moves of the game are implemented as an enumeration. The options are:
  - MOVE.UP
  - MOVE.RIGHT
  - MOVE.DOWN
  - MOVE.LEFT
  - MOVE.NEUTRAL
- Ms. Pac-Man controllers need to return one of these moves at every time step.
- If the controller returns null or an illegal move (e.g., trying to go up even though there is a wall), the game engine will try to play the last move if possible or else choose a legal move randomly.

Your first pacman Controller !

- Make the new class inherited from pacman.Controller
- Override the getMove() method

- The class should look like this by now:

```
package pacman.controllers.examples;
import pacman.controllers.Controller;
import pacman.game.Game;

public class MyFirstPacman  extends Controller{

@Override
public Object getMove (Game game, long timeDue) {
// TODO Auto-generated method stub
      return null;
}
}
```

Add code for a random controller

- getMove() method has 40 ms to compete and return the next move to be made by pacman

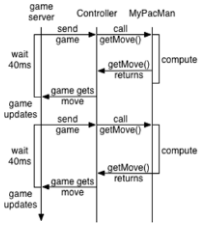
```
public class MyFirstPacman  extends Controller
{
    Random rand = new Random();
    @Override
    public Object getMove(Game game, long timeDue) {
          MOVE[] possibleMoves =
          game.getPossibleMoves(game.getPacmanCurrentNodeIndex());
          return possibleMoves[rand.nextInt(possibleMoves.length)];
    }

}
```

Framework

- The game is played asynchronously in real time: every discrete game tick, the game uses the moves supplied by the controllers to update the gamestate.
- At each game tick, the controllers have 40 ms to respond.
- The controllers are given a copy of the current game and the time the move is due. Each controller can then query the game using its many methods to compute an appropriate response. It is the responsibility of the controller to respond in a timely manner (this is one of the challenges of the competition).
- If a controller returns a move on time, the game will use that move to advance the game.
- If a controller does not replay on time, the game tries to play the previous move or, if that is not possible, chooses a legal move randomly.

Framework

- The diagram below illustrates how the temporal aspects of the game is implemented.



Game State

- The game object is a copy of the current game state. Some sample methods to retrieve game states are:
  - game.getPacmanCurrentNodeIndex()
  - game.getActivePowerPillsIndices()
  - game.getActivePillsIndices()
  - game.getGhostCurrentNodeIndex(GHOST.PINKY)
  - game.isGhostEdible(GHOST.PINKY)
  - ……
  - and many more for you to explore

Utility functions

- There are several other utility functions too that are exposed via game object. For example,
  - game.getNextMoveTowardsTarget(…)
  - game.getNextMoveAwayFromTarget(…)
  - game.getClosestNodeIndexFromNodeIndex(…)
  - game.getEuclideanDistance(…)
  - and many more

Distance Measures

- Manhattan Distance:
  - The standard heuristic for a square grid is the Manhattan distance.

    Manhattan distance between (x1,y1) and (x2,y2) is:

    (abs(x1- x2) + abs(y1 – y2))
- Euclidean Distance:
  - Straight line distance
  - Manhattan distance between (x1,y1) and (x2,y2) is Sqrt((x1- x2)^2 + (y1 – y2)^2)

Refine your controller

- Go to the nearest power pill and consume it

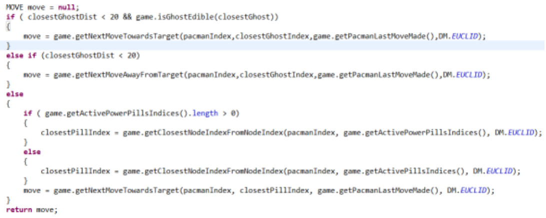
```
public class MyFirstPacman  extends Controller{
@Override
        public Object getMove(Game game, long timeDue)
        {
                // TODO Auto-generated method stub
                int pacmanIndex = game.getPacmanCurrentNodeIndex();
                int closestPillIndex =
                game.getClosestNodeIndexFromNodeIndex(pacmanIndex,
                game.getActivePowerPillsIndices(), DM.EUCLID);

                MOVE move = game.getNextMoveTowardsTarget(pacmanIndex,
                closestPillIndex, game.getPacmanLastMoveMade(), DM.EUCLID);
                return move;
        }
}
```

Further refinement

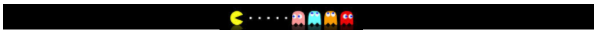
- Done with power pills? ………Consume remaining pills too to cross the level

```
public class MyFirstPacman  extends Controller{
        @Override
        public Object getMove(Game game, long timeDue) {
                int pacmanIndex = game.getPacmanCurrentNodeIndex();
                int closestPillIndex = -1;

                if ( game.getActivePowerPillsIndices().length > 0)
                {
                        closestPillIndex = game.getClosestNodeIndexFromNodeIndex(pacmanIndex,
                        game.getActivePowerPillsIndices(), DM.EUCLID);
                }
                else
                {
                        closestPillIndex = game.getClosestNodeIndexFromNodeIndex(pacmanIndex,
                        game.getActivePillsIndices(), DM.EUCLID);
                }
                MOVE move = game.getNextMoveTowardsTarget(pacmanIndex, closestPillIndex,
                game.getPacmanLastMoveMade(), DM.EUCLID);
                return move;
        }
}
```

Beware of ghosts !



```
MOVE move = null;
if ( closestGhostDist < 20 && game.isGhostEdible(closestGhost))
{
    move = game.getNextMoveTowardsTarget(pacmanIndex,closestGhostIndex,game.getPacmanLastMoveMade(),DM.EUCLID);
}
else if (closestGhostDist < 20)
{
    move = game.getNextMoveAwayFromTarget(pacmanIndex,closestGhostIndex,game.getPacmanLastMoveMade(),DM.EUCLID);
}
else
{
    if ( game.getActivePowerPillsIndices().length > 0)
    {
        closestPillIndex = game.getClosestNodeIndexFromNodeIndex(pacmanIndex, game.getActivePowerPillsIndices(), DM.EUCLID);
    }
    else
    {
        closestPillIndex = game.getClosestNodeIndexFromNodeIndex(pacmanIndex, game.getActivePillsIndices(), DM.EUCLID);
    }
    move = game.getNextMoveTowardsTarget(pacmanIndex, closestPillIndex, game.getPacmanLastMoveMade(), DM.EUCLID);
}
return move;
```



What else?

Come up with your best pacman controller…..We’ll have a competition !