Ms. Pacman

DA272A – Artificial Intelligence

Malmö University

Faculty of Technology and Society

Jose Font– jose.font@mah.se

MALMÖ HÖGSKOLA

INTRODUCTION

- Opensource version of the classic.
- Framework written in Java, created by Philipp Rohlfshagen, David Robles, and Simon Lucas (University of Essex, UK).
- Used frequently in the Pacman VS Ghosts competition during many videogame AI international conferences.
- Can be played both by a human player or a computer controller.
- Custom controllers for Ms Pacman and the Ghosts (Pinky, Inky, Blikny, and Sue) can be easily added to the game.

INSTALLATION

- Get Eclipse.
- Download and unzip Ms_Pacman.rar.
- Import the Project in the unzipped folder from Eclipse:
  - File > Import > Existing Projects into Workspace

- pacman.Executor is the entry point.
- The framework can be run in several modes. Uncomment one of them in the main method:
  - **exec.runGameTimed(new HumanController(new KeyBoardInput()),new StarterGhosts(), true);**

  Run the game in visual mode, so that the human player plays as Ms Pacman using the keyboard, and the ghosts are controlled by a basic AI program (StarterGhosts).

  - **exec.runGameTimed(new StarterPacMan(),new StarterGhosts(), true);**

  Run the game in visual mode, so that both, Ms Pacman and the ghosts, are controlled by a basic AI programs (StarterPacMan and StarterGhosts).

  - **exec.runGameTimed(new DataCollectorController(new KeyBoardInput()),new StarterGhosts(), true);**

  Run the game in visual mode, so that the human player plays as Ms Pacman using the keyboard, and the ghosts are controlled by a basic AI program (StarterGhosts). **Record gameplay data into the file "myData/trainingData.txt".**

- pacman.Executor is the entry point.
- The framework can be run in several modes. Uncomment one of them in the main method:
  - **exec.runExperiment(new RandomPacMan(),new RandomGhosts(),numTrials);**

  Run the game without graphics, with two random controllers for both Ms. Pacman and the Ghosts. The game is played **numTrials** times, returning the score that Ms Pacman got in every playout, as well as the average score.
- The controllers for Ms Pacman and the Ghosts are fully customizable. Some interesting examples can be found in the package **pacman.controllers.examples.**

CODING A CONTROLLER FOR MS PACMAN

- A valid controller inputs the current game state at time t, and returns one of the following commands for Ms. Pacman: UP, LEFT, RIGHT, DOWN o NEUTRAL. Ms. Pacman will make this move at t+1.
- A controller is a class that implements the interface Controller<MOVE>, overriding getMove.

```
public class StarterPacMan extends Controller<MOVE>
{

    public MOVE getMove(Game game,long timeDue)
    {
```

- getMove gets the game state as an input argument, as well as the maximum time allowed for a response.
- getMove returns one of the valid moves contained in the MOVE enum: {UP, LEFT, RIGHT, DOWN, NEUTRAL}
- getMove is a callback called automatically by the game at every time t.

- The game object contains all the information needed about the game state to code an intelligent agent for Ms. Pacman. It also provides simple ways to retrieve that information:
  - Maze m = game.getCurrentMaze(); returns the current maze.
  - Node[] graph = m.graph; returns all the nodes in the current maze as a Node array.
  - intpacmanPos= game.getPacmanCurrentNodeIndex(); returns the index of Pacman's current node.
  - int[] activePills=game.getActivePillsIndices(); returns the indices for all the nodes containing pills.
  - int[] activePowerPills=game.getActivePowerPillsIndices(); returns the índices for all the nodes containing power pills.
  - public int getShortestPathDistance(int fromNodeIndex, int toNodeIndex); returns the shortest distances from "fromNodeIndex" to "toNodeIndex".

- The game object contains all the information needed about the game state to code an intelligent agent for Ms. Pacman. It also provides simple ways to retrieve that information:
  - game.getNextMoveTowardsTarget(int fromNodeIndex,int toNodeIndex,DM distanceMeasure); returns the move to the next node in the way from “fromNodeIndex” to “toNodeIndex”, following one of the implemented distance measures: {PATH, EUCLID, MANHATTAN}.
  - getNextMoveAwayFromTarget(int fromNodeIndex, int toNodeIndex, DM distanceMeasure); returns the opposite move to the previous one.
  - int getGhostCurrentNodeIndex(GHOST ghostType); return the index for the node where “ghostType” is, where “ghostType” can be: {BLINKY, PINKY, INKY, SUE}.
  - int getGhostEdibleTime(GHOST ghostType); returns for how long “ghostType” will still be edible.
  - int getGhostLairTime(GHOST ghostType); returns for how long will “ghostType” be trapped inside the lair.

CODING A CONTROLLER FOR THE GHOSTS

- A valid controller inputs the current game state at time t, and returns one of the following commands **for each of the ghosts**: UP, LEFT, RIGHT, DOWN o NEUTRAL. Each ghost will make its corresponding move at t+1.
- A controller is a class that implements the interface ***Controller<EnumMap<GHOST,MOVE>>,*** and overrides ***getMove***.
- ***getMove*** returns an ***EnumMap<GHOST,MOVE>***, i. e., a list that contains a move (**UP, LEFT, RIGHT, DOWN, NEUTRAL)** for each of the ghosts**.**
- getMove is a callback called automatically by the game at every time t.

RUNNING CUSTOM CONTROLLERS

- To play the game using custom controllers, use instances of those controllers in the appropriate method in Executor.
  - exec.runGameTimed(**new MiControladorPacMan()**,**new MiControladorGhosts()**, true);