Ms. Pac-Man AI

Overview

This is a version of Ms. Pac-Man that closely represents the original game. The representation has a few minor changes to make writing the AI for both Ms. Pac-Man and the Ghosts fair.

You are given the controller interfaces in Java that allow you to make the decisions for either Ms. Pac-Man Agent or all the Ghost Agents. The rest of this document explains the basics of the decision making AI that you are to implement for this project.

***** IMPORTANT API NOTE *****

An *index* is **NOT** the same as a *node*! The documentation occasionally confuses the two, but we will make a clear distinction here. *Indices* are only used for the check*() functions, have values from 0 to (numOfItems - 1), and are often represented by "i" in a for() loop. *Nodes* are used by the pathfinding functions, represent physical locations, have random-looking values provided by the Ms. Pac-Man application that are ONLY used for identification, and will often be represented by something like "nodes[i]" instead. Look at /05 – Ms. Pac-Man/Workspace/System/src/game/controllers/examples/NearestPillPacMan.java for an example of how to implement the difference properly. If you're ever in doubt about whether you have indices or nodes at any given point in your code, fire up your debugger and inspect them directly.

Game Mechanics

Below describes the goals of either Ms. Pac-Man or the Ghost Team and the main game mechanics that are the deciding factors in them achieving their goal.

Ms. Pac-Man Goals

The main goal of the game is for Ms. Pac-Man to achieve the highest score possible. She does this by eating various things in the game levels. These levels are represented by the maze that has no dead ends. Ms. Pac-Man can eat either pills. Power pills, or ghosts. The pills are represented by small dots while the power pills are larger. Eating a small pill awards Ms. Pac-Man 10pts. Eating a power pill awards her 50pts and gives Ms. Pac-Man the ability to eat ghosts for a short period of time. Each ghost eaten awards 200 * 2^(x-1) points where x is the number of the ghost eaten after the Power Pill. Ms. Pac-Man starts with 3 lives and can continue to eat until she runs out of lives.

Ghost Team Goals

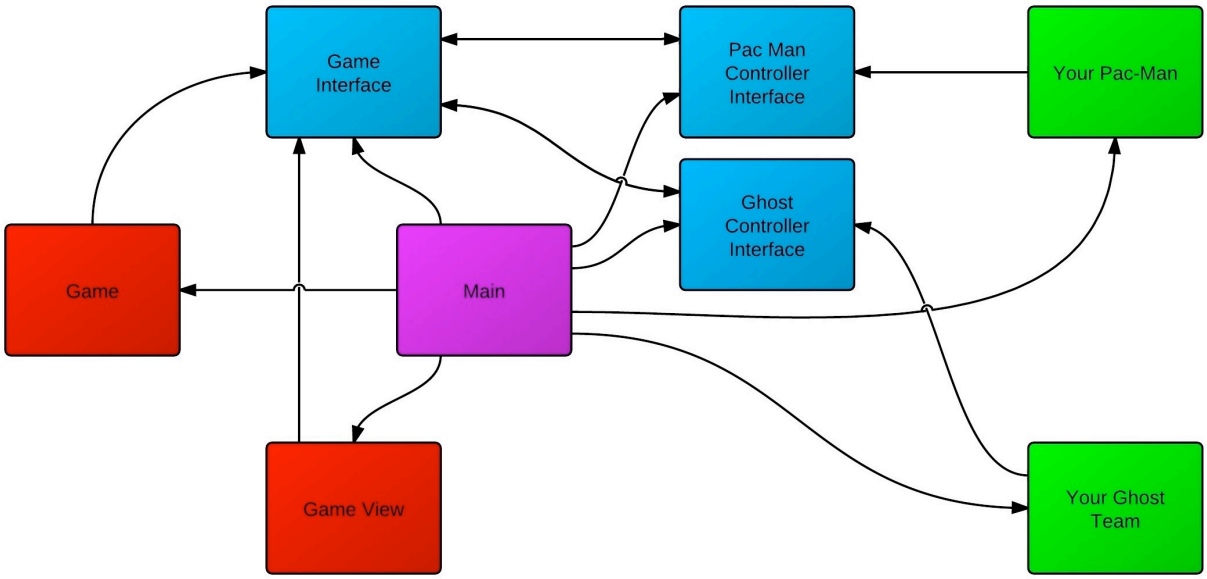
As you might guess the goal of the ghost team AI is to limit the score of Ms. Pac-Man as much as possible during her lives. There are 4 Ghosts in total. The quickest way to limit Ms. Pac-Man's score is to catch her with the ghosts while she is not under the effects of the Power Pills. Once caught Ms. Pac-Man loses a life.

Additional Information

- Ms. Pac-Man gains a life every 10000 points.
- There are 4 Mazes A B C D. They cycle in that order.
- The further the level the shorter the effects of the Power Pills last on Ms. Pac-Man.
- Eating another Power Pill while Ms. Pac-Man is already under the effects of another Power pill will reset the timer, but will also reset the x in the score equation for eating ghosts explained above.
- Both Ms. Pac-Man and the Ghosts move at a constant speed.
- Ghosts can never reverse their direction while Ms. Pac-Man can.
- There is a very small random global reversal element internal to the ghosts. The Ghost AI controller cannot control this. If it is successful. All ghosts will reverse direction.
- In the rare case a given level lasts 3000 turns the rest of the pills are awarded to Ms. Pac-Man and the next level starts.

System API

You are given an interface to a game that functions in full game ticks. Ms. Pac-Man and the Ghost Team are on separate threads and are given 40ms. To return their actions (direction changes). All 4 ghosts are part of this team and they must share the 40ms between them to decide what they are going to do. If no action is returned within the 40ms the thread function kicks out and the previous action returned is repeated.



Above shows a class diagram that represents major systems of this API. You are responsible for writing java code inside the Your Pac-Man class. This class derives off the Pac-Man Controller Interface. It overrides the getAction method of this base. This method takes in a copy of the game via a Game Interface. This base game class gives access to all the relevant information about the correct state of the game that you will be determining your return upon. To give you an idea of how the API runs, Main starts the Game and Your Pac-Man and runs the game calling your codes getAction with a Pac Man Controller Interface the represents your class. It only cares about the return of this function. The Game View class is only in this diagram because you will be running your AI in two modes. Visual mode where the Game View is used to render the game and Test mode where 100 rounds are run as fast as possible to give you an accumulative score that you can divide by 100 and determine your AI's average.

PacManController

This java class represents the interface the game uses for Ms. Pac-Man. Your implementation derives from this interface.

Relevant Methods

```
public int getAction(Game game,long timeDue);
```

Provided a Game State that has information about Ms. Pac-Man, Ghosts, Nodes, and the Maze. The AI must return a direction for Ms. Pac-Man to travel in the form of an integer. Returns a direction in the form of an integer. Valid returns are UP(0) RIGHT(1) DOWN(2) LEFT(3).

GhostController

This java class represents the interface the game uses for the ghost team. Your implementation derives from this interface.

Relevant Methods

`public int[] getActions(Game game,long timeDue);`

Provided a Game State that has information about Ms. Pac-Man, Ghosts, Nodes, and the Maze. The AI must return a direction for Ms. Pac-Man to travel in the form of an integer. Returns an array of integers that represent the direction for each of the 4 ghosts. Valid returns for each int are UP(0) RIGHT(1) DOWN(2) LEFT(3).

Game

This class represents a snapshot of the game information at a certain point in time. This is passed into the get action functions for the controllers. It contains information such as the maze, pills and ghosts. The methods descriptions bellow are also in the comments for the game java class.

Relevant Methods

`public Game copy();`

returns an exact copy of the game (forward model)

`public int[] advanceGame(int pacManDir,int[] ghostDirs);`

advances the game using the actions (directions) supplied; returns all directions played [PacMan, Ghost1, Ghost2, Ghost3, Ghost4]

`public int getReverse(int direction);`

returns the reverse of the direction supplied

`public boolean gameOver();`

returns true is Ms Pac-Man has lost all her lives or if MAX_LEVELS has been reached

`public boolean checkPill(int pillIndex);`

checks if the pill specified is still available

`public boolean checkPowerPill(int powerPillIndex);`

checks if the power pill specified is still available

`public int[] getPacManNeighbours();`

returns an array of size 4, indicating neighbouring nodes for the current position of Ms Pac-Man. E.g, [-1,12,-1,44] for neighbours 12 and 44 in direction RIGHT and LEFT

`public int[] getGhostNeighbours(int whichGhost);`

returns an array of size 4, indicating neighbouring nodes for the current position of the ghost specified. Replaces the direction corresponding to the opposite previous direction with -1

`public int getCurLevel();`

returns the current level

`public int getCurMaze();`

returns the current maze

`public int getCurPacManLoc();`

returns the node index Ms Pac-Man is at

`public int getCurPacManDir();`

returns the last direction taken by Ms Pac-Man

`public int getLivesRemaining();`

returns the number of lives remaining for Ms Pac-Man

`public int getCurGhostLoc(int whichGhost);`

returns the node index for the ghost specified

`public int getCurGhostDir(int whichGhost);`

returns the last direction taken by the ghost specified

`public int getEdibleTime(int whichGhost);`

returns the edible time (time left in which the ghost can be eaten) for the ghost specified

`public boolean isEdible(int whichGhost);`

returns true if the ghost is currently edible

```
public int getScore();
```

returns the score of the game

```
public int getLevelTime();
```

returns the time for which the CURRENT level has been played

```
public int getTotalTime();
```

returns the time for which the game has been played (across all levels)

```
public int getNumberPills();
```

returns the total number of pills in this maze (at the beginning of the level)

```
public int getNumberPowerPills();
```

returns the total number of power pills in this maze (at the beginning of the level)

```
public int getLairTime(int whichGhost);
```

returns the time remaining the ghost specified spends in the lair

```
public boolean ghostRequiresAction(int whichGhost);
```

returns true of ghost is at a junction and a direction is needed

```
public String getName();
```

returns the name of the maze

```
public int getInitialPacPosition();
```

returns the position where Ms Pac-Man starts at the beginning of the level

```
public int getInitialGhostsPosition();
```

returns the position where the ghosts starts at the beginning of the level, AFTER leaving the lair

```
public int getNumberOfNodes();
```

returns the total number of nodes in the graph (pills, power pills and empty)

```
public int getX(int nodeIndex);
```

returns the x-coordinate of the node specified

```
public int getY(int nodeIndex);
```

returns the y-coordinate of the node specified

```
public int getPillIndex(int nodeIndex);
```

returns the pill index of the node specified (can be used with the bitset for the pills)

```
public int getPowerPillIndex(int nodeIndex);
```

returns the power pill index of the node specified (can be used with the bitset for the power pills)

```
public int getNeighbour(int nodeIndex,int direction);
```

returns the neighbour of the node specified for the direction supplied

```
public int[] getPillIndices();
```

returns indices to all nodes with pills

```
public int[] getPowerPillIndices();
```

returns indices to all nodes with power pills

```
public int[] getJunctionIndices();
```

returns indices to all nodes that are junctions

```
public boolean isJunction(int nodeIndex);
```

returns true if node is a junction (more than 2 neighbours)

```
public int getNumNeighbours(int nodeIndex);
```

returns the number of neighbours of the node specified

```
public enum DM{PATH,EUCLID,MANHATTEN};
```

simple enumeration for use with the direction methods (below)

```
public int getNextPacManDir(int to,boolean closer,DM measure);
```

returns the direction Ms Pac-Man should take to approach/retreat from the node specified, using the distance measure specified

```
public int getNextGhostDir(int whichGhost,int to,boolean closer,DM measure);
```

returns the direction the ghost specified should take to approach/retreat from the node specified, using the distance measure specified

```
public int getPathDistance(int from,int to);
```

returns the shortest path distance (Dijkstra) from one node to another

```
public double getEuclideanDistance(int from,int to);
```

returns the Euclidean distance between two nodes

```
public int getManhattenDistance(int from,int to);
```

returns the Manhatten distance between two nodes

```
public int[] getPossiblePacManDirs(boolean includeReverse);
```

returns the set of possible directions for Ms Pac-Man, with or without the direction opposite to the last direction taken

```
public int[] getPossibleGhostDirs(int whichGhost);
```

returns the set of possible directions for the ghost specified (excludes the opposite of the previous direction)

```
public int[] getPath(int from,int to);
```

returns the path from one node to another (e.g., [1,2,5,7,9] for 1 to 9)

```
public int[] getGhostPath(int whichGhost,int to);
```

returns the path from one node to another, taking into account that reversals are not possible

```
public int getTarget(int from,int[] targets,boolean nearest,DM measure);
```

selects a target from 'targets' given current position ('from'), a distance measure and whether it should be the point closest or farthest

```
public int getGhostTarget(int from,int[] targets,boolean nearest);
```

selects a target for a ghost (accounts for the fact that ghosts may not reverse)

```
public int getGhostPathDistance(int whichGhost,int to);
```

returns the distance of a path for the ghost specified (accounts for the fact that ghosts may not reverse)

GameView

This class is responsible for the rendering of Ms. Pac-Man there are a few static utility functions that you can use for debugging purposes. These methods include addPoints and addLines.

Samples

There are sample AI's with usage of many of the methods listed above included within the systems game controller examples.

Grading

You will be graded based on your score after 100 game rounds of your Ms. Pac-Man AI against the default ghost team AI in test mode. Give it a second to complete the 100 rounds before it will spit out the results in the outputs window. You can switch between the Visual or Test run configurations by clicking the down arrow next to the Debug or Run buttons. Your goal is to match or exceed the example Ms. Pac-Man score in the same test run of 100 rounds. If you exceed the Full Sail AI's score, each 1000 points over will give you 1% extra credit (but only in increments of 1%). This extra credit is capped at 10%.

Submissions

You will upload a compressed (Zipped) file named *<lastName>.<firstName>.MsPacmanLab.zip* which should include:

MsPacManAgent.java

Also, please submit any source/header files you have created yourself.

Place this file in the ROOT of your zip file, not in a folder or subfolder. DO NOT SUBMIT EXECUTABLES, LIBRARIES, OR OBJECT FILES.