

ICE Pescape

Yoshitaka Kashifuji, Junichi Oda, Hiroshi Matsumoto, Daichi Hirono, and Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Department of Human and Computer Intelligent, Ritsumeikan University
ruck@ci.ritsumeai.ac.jp
23 May 2008

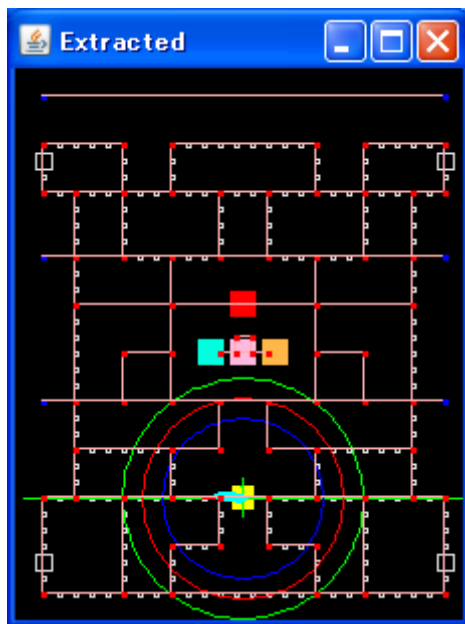
1. Introduction

This is our Ms Pac-man software controller in Java for the WCCI 2008 contest. Its name is ICE Pescape and developed based on the sample controller (pac.zip) available from the site below:

<http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html#Introduction>

ICE Pescape is aimed at achieving moderate scores on a low performance machine. On a high performance machine, a more aggressive version called ICE Pambush might obtain much higher scores. A demo video clip of ICE Pescape is available at the site below:

<http://jp.youtube.com/watch?v=nSJJ3B2lvIc>



ICE Pescape uses the same image processing as the aforementioned sample controller to extract all relevant objects: Ghosts, Pills, Power-Pills, and Ms PacMan. The left figure is an example of the window titled "Extracted" that shows how such objects are perceived. This window usually appears on the top left of the display. As shown in this figure, all corners and cross sections in the Ms PacMan window are represented by small red nodes, their connected paths by pink links, ghosts by large squares in their original colors, pills by small white squares, power pills by large white squares and Ms. PacMan by the yellow square.

Our Ms PacMan moves along those links towards the target location. We describe in Sec. 3 how to determine such a target and what is the meaning of red, blue, and green circles surrounding Ms PacMan.

2. Setting and Running

ICE Pescape can be run by moving to /src/ and executing `java pacman/MsPacInterface`.

If the current parameter setting is correct, the "Extracted" window will have the content similar to that shown in Fig. 1; otherwise, parameters in **MsPacInterface.java** (in /src/pacman/) must be changed accordingly. Below are our hints on which parameters to try first!

Currently, all related parameters follow the recipe in the sample controller, i.e.,

```
/*Positions of the Ms PacMan window*/
static int left = 530;
static int top = 274;
/*Colors*/
static int blinky = -65536;
static int pinky = -18689;
static int inky = -16711681;
static int sue = -18859;
static int pacMan = -256;
static int edible = -14408449;
static int pill = -2434305;
```

However, at our environment we must change some of them (those in italic below) in order to properly run the controller as follows:

```
/*Positions of the Ms PacMan window*/
static int left = 515;
static int top = 160;
/*Colors*/
static int blinky = -65536;
static int pinky = -18210;
static int inky = -16711714;
static int sue = -18361;
static int pacMan = -256;
static int edible = -14605858;
static int pill = -2171170;
```

Those in italic are worth trying first. For colors, we recommend you use the method `getRGB` in the `BufferedImage` class to derive the object colors actually displayed in the Ms PacMan window on your screen. To recompile (JDK 1.6) the program, execute `javac pacman/MsPacInterface.java`. Hope this helps!

3. Controller

The A* algorithm is used to find the shortest path, the one with the lowest cost, between Ms PacMan and the target location. At each iteration, a node cost is assigned to each node and is added to the distance cost in A*. For distance computation, the Manhattan distance is used, and our cost assignment is as follows:

- The node nearest to a ghost:
1000*(1000 — [the distance between the ghost position and the node])
- Four neighbor (top, bottom, left, and right) nodes of a node of the above type:
1000*(1000 — the distance between the node nearest to a ghost and the neighbor node)
- Nodes with no pill:
100

At each iteration, one of the following five rules (in decreasing priority) is fired.

Rule 1: If a ghost enters the red-circle boundary and at least one power pill lies in the green-circle boundary, then target the nearest power pill in the green-circle boundary.

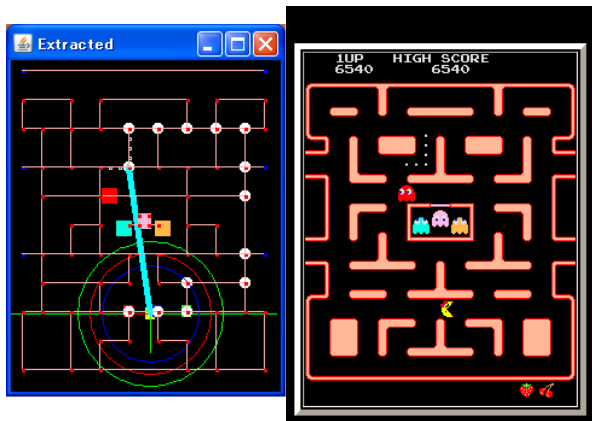
Rule 2: If a ghost enters the red-circle boundary, then target one of the four outmost-corner nodes that is furthest to the four ghosts.

Rule 3: If an edible ghost enters the blue-circle boundary, then target it.

Rule 4: Target the nearest pill

Rule 5: If A* cannot produce any path, use the same rule as that in the sample controller.

Note that these rules are the same rules used in ICE Pambush, except that two rules related to ambushing in ICE Pambush are not available, making ICE Pescape less aggressive than its sister controller. However, ICE Pescape requires less CPU time, making it performs better than ICE Pambush on a lower performance machine.



The left figures show a situation where Rule 6 is fired. Note that Ms PacMan takes a roundabout way from the ghosts according to the resulting path from A*, depicted by white circle nodes in the "Extracted" window.

4. Results

In average, ICE Pescape scores above 5,000 on Sony Vaio S-type (Core 2 Duo, 2.4 GHz, 2 GB memory) in the stamina-battery mode with other programs running to increase the load for simulating a situation of a low-performance machine.

Acknowledgement

Special thanks go to Chota Tokuyama and Ko Oda for their helps in the beginning phase and the testing phase of development, respectively.