

ICE Pambush 3

Hiroshi Matsumoto, Takashi Ashida, Yuta Ozasa, Takashi Maruyama, and Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Department of Human and Computer Intelligent, Ritsumeikan University
ruck [at] ci.ritsumeik.ac.jp
30 August 2009

1. Introduction

This is our Ms. Pac-Man software controller in Java for the IEEE CIG 2009 contest. Its name is ICE Pambush 3. ICE Pambush 3 is an improved version of ICE Pambush 2, the winner of the IEEE CEC 2009 contest. Major improvements are as follows:

1. Automatic detection of the game map location, leading to no need for cumbersome adjustment of the game window page
2. Use of Dijkstra distance between any two nodes, giving us more precise distance information
3. New effective rules for luring ghosts and eating pills, enabling achievement of high scores.

The above figure shows the histogram of its scores with the average of 16,000 during the last 50 games before submission on a Sony Vaio Laptop (Stamina Mode, VGN-SZ, Intel Core Duo CPU 2.40 GHz, 2 GB Memory, Vista Ultimate). We tested the controller with the Web-version of Ms. Pac-Man at the following site:

<http://www.webpacman.com/Ms.Pac-Man.htm>

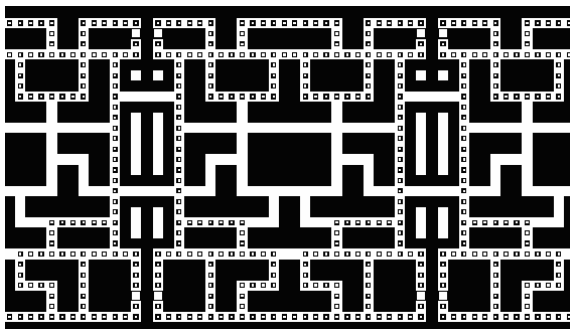
A demo video clip of ICE Pambush 3 is available at the site below:

<http://www.youtube.com/watch?v=Zo0YujjX1PI>

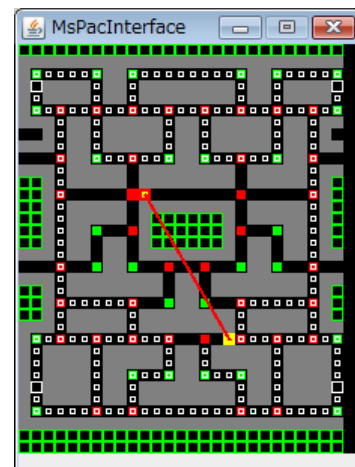
Hope that similar performance will be achieved at the contest!!

The right figure shows our representation of the game. This window, titled MsPacInterface, usually appears on the top left of the display. There, all pills are in white boxes and power pills in larger white boxes. Each corner and cross point are superimposed by green and red, respectively. A ghost is represented by its own color, and our Ms. Pac-Man is represented by yellow. The red line points from Ms. Pac-Man to the nearest non-edible ghost.

Our Ms. Pac-Man moves along the black paths towards the target location. As done in ICE Pambush 2, to exploit warp points, the game map internally used in our controller is an extended



version of the one shown in the



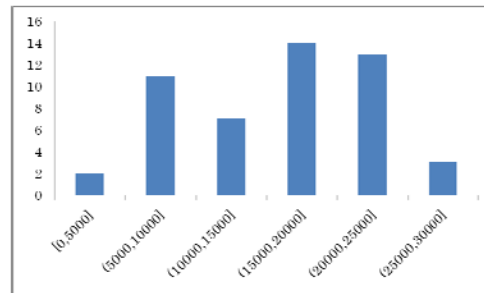
MsPacInterface window. This map concatenates the right half + the whole map + the left half (c.f. the left figure). With this map, our Ms. Pac-Man will go through a warp point if the target is located near the opposite warp point.

2. Set-Up and Execution

After launching the Web browser page of Ms. Pac-Man(www.webpacman.com/Ms.Pac-Man.htm),

ICE Pambush 3 can be run by moving to /src/ and executing

```
java main/MsPacInterface.
```



Because of implementation of automatic detection of the game map in the Web browser page, it is no longer necessary to adjust the position of the game window. The program will automatically handle this issue. However, **please do not change the position of the game window during execution of the game.** After inserting a coin and starting the game, if the current parameter setting is correct, the MsPacInterface window will have the content similar to the second figure in page one, where all objects are shown in the colors described above; our controller should easily earn more than 5000 scores. In summary, the three figures below summarize from left to right the procedure for executing our controller.



However, if ICE Pambush 3 scores badly, say, below 5000, for a number of consecutive games, color setting might need adjustment. For this, the corresponding color related parameters in **MsPacInterface.java** (in /src/main/) must be changed accordingly. Below are our hints on which parameters to try first! Currently, all color related parameters used at our environment during development (**the Web version** of Ms. Pac-Man using **Internet Explorer**) are as follows:

```

blinky = -65536;           //blinky
pinky = -18210;           //pinky = -18689 in the sample controller
inky = -16711714;        //inky=-16711681 in the sample
sue = -18361;            // sue = -18859 in the sample
pacMan = -256;
edible = -14605858;      //edible ghost = -14408449 in the sample
pill1 = -2171170;        //pill at the first stage = -2434305 in the sample
pill2 = -256;            //pill color at the second stage
pill3 = - 65536;         //pill color at stage 3
pill4 = -2171170;        //pill color at stage 4
back = -16777216;        // color for the background

```

Please try first to modify **those parameters in red**. After modifying those parameters, please recompile (JDK 1.6) the program by

```
javac main/MsPacInterface.java
```

Hope this helps!

3. Controller

Our controller moves Ms. Pac-Man based on path costs. We adopted two variants of the A* algorithm to find the best path, the one with the lowest cost, between Ms. Pac-Man and the target location. The Dijkstra distances between all pairs of nodes were computed and stored in the related files in advance. At the beginning of each game, the distance information is loaded and used in our search algorithm. At each iteration, one of the following nine rules (in decreasing priority) is fired.

#Rule 1:

IF (distance(nearest_power_pill) \leq 5(3*)) AND (4 \leq distance(nearest_ghost) \leq 8) AND (distance(ghost_nearest_to_the_nearest_power_pill) \geq 6(4*)),

THEN stop moving and ambush (enter the ambush state) at the corner or cross point near the nearest power pill waiting for a ghost to come closer,

where distance(nearest_power_pill) is the distance from Ms. Pac-Man to the nearest power pill, distance(nearest_ghost) the distance from Ms. Pac-Man to the nearest ghost, and distance(ghost_nearest_to_the_nearest_power_pill) the distance from Ms. Pac-Man to the ghost nearest to the power pill nearest to Ms. Pac-Man, and the numbers with * in the parentheses are those for the second stage of the game.

#Rule 2:

IF at least one power pill exists AND no edible ghost exists AND Ms. Pac-Man is at the ambush state AND ((distance(nearest_ghost) \leq 4) OR (distance(ghost_nearest_to_the_nearest_power_pill) \leq 6(4*))),

THEN move to the nearest power pill with the A* version two.

#Rule 3:
IF at least one power pill exists AND no edible ghost exists AND
(distance(nearest_ghost) \leq 8) AND (distance(nearest_power_pill) \leq 4),
THEN move to the nearest power pill with the A* version two.

#Rule 4:
IF at least one power pill exists AND no edible ghost exists AND
(distance(nearest_ghost) \leq 8),
THEN move to the nearest power pill with the A* version one.

#Rule 5:
IF at least one edible ghost exists AND
(distance(nearest_ghost) \leq 8) AND (distance(nearest_edible_ghost) \leq 10),
THEN move to the nearest edible ghost with the A* version one,
where distance(nearest_edible_ghost) is the distance from Ms. Pac-Man to the nearest edible ghost.

#Rule 6:
IF at least one pill exists AND
(distance(nearest_ghost) \leq 8),
THEN move to the nearest pill with the A* version one.

#Rule 7:
IF at least one edible ghost exists AND
(distance(nearest_ghost) \geq 9) AND (distance(nearest_edible_ghost) \leq 10),
THEN move to the nearest edible ghost with the A* version two.

#Rule 8:
IF at least one pill exists AND (no pill# exists OR
((distance(nearest_ghost) \geq 9) AND (distance(nearest_power_pill) \geq 20) AND (distance(pill) \leq 10))),
THEN move to the nearest pill with the A* version two,
where pill# indicate those pills not lying between a pair of two cross points whose connected path contains a power pill.

#Rule 9:
IF at least one pill exists AND
(distance(nearest_ghost) \geq 9),
THEN move to the nearest pill# with the A* version two.

To find a path, the A* version one considers the distance cost, the ghost cost, and the corner cost while the A* version two considers only the distance cost, where cost definitions are given below. The former version of A* is used in more critical situations, such as when a ghost is nearby, than the latter one. In addition, the depth of search is also different between these two versions. The n depth indicates that the search space covers all paths from the current grid of Ms. Pac-Man to the n th corner or cross point in the same quarter of the target location, with the constraint that the distance from Ms. Pac-Man to the i th corner or cross point is always larger than that of the $i-1$ th one. In particular, the search depth of the A* version one and two is 5 and 10, respectively.

Costs are defined such that Ms. Pac-Man can manage to reach the target location without being hit by a ghost. The definition of each cost is given below, and the costs below are accumulated for the corresponding corner or cross point.

#Distance Cost at point X:
= [distance(X) + distance(X_to_target) – distance(target)]*1000,
where X is the i th-level search corner or cross point from Ms. Pac-Man, $i = 1$ to 5 (A* version one) or 1 to 10 (A* version two), distance(X) is the distance from Ms. Pac-Man to X, distance(X_to_target) is the distance from X to the target location, and distance(target) is the distance from Ms. Pac-Man to the target location.

#Ghost Cost I at point X away from ghost Y:
= 500,000/[distance(Y_to_X)^2],
where X = the i th corner or cross point from ghost Y, $i = 1$ and 2, and distance(Y_to_X) is the distance from ghost Y to point X.

Ghost Cost II at a corner or cross point where there is a ghost on it:
= 500,000

#Ghost Cost III at a corner or cross point behind a ghost moving on the same straight path as Ms. Pac-Man:
= 2,000,000

Corner Cost at each corner:
= 5,000

END