ICE Pambush 4
Takashi Ashida, Takeru Miyama, Hiroshi Matsumoto, and Ruck Thawonmas
Intelligent Computer Entertainment Laboratory
Department of Human and Computer Intelligent, Ritsumeikan University, Japan
ruck [at] ci.ritsumei.ac.jp
15 August 2010

1. Introduction
This is our Ms. Pac-Man software controller in Java for the IEEE CIG 2010 contest. Its name is ICE Pambush 4. ICE Pambush 4 is an improved version of ICE Pambush 3, the winner of the IEEE CIG 2009 contest. Major improvements are as follows:
1. All distances concerning the four ghosts are computed considering the ghost directions. For example, the distance between Ms. Pac-Man and Pinky is the Dijkstra distance between Ms. Pac-Man and the node facing and nearest to Pinky plus the distance between this node and Pinky.
2. The extended (concatenated) internal map, used in ICE Pambush 3, is no longer used here. The internal map used in the ICE Pambush 4 has the same size as the current game map.
3. Hand tuning of the distance parameters in the 10 rules, described below, was conducted based on parameter optimization results using Evolutionary Strategy [1].

ICE Pambush 4 achieved the mean score of 18,561 (median of 18,330) and the maximum score of 38,910 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). Although we have 15 licenses of the Microsoft Revenge of Arcade, we developed and tested this version of 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 4 is available at the site below:
http://www.youtube.com/watch?v=8cFChSC6-6w

In this video clip, the window MsPacInterface 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 that might be a possible threat. Ms. Pac-Man moves along one of the black paths towards the target location.
A known problem is that our fast object recognition algorithm cannot distinguish Ms. Pac-Man and the banana item; their main color is yellow. So right after the banana item appears in the game, Ms. Pac-Man will stop moving and hence will be easily eaten by a ghost. This situation should occur when the score is more than 34,000 or so.

2. Set-Up and Execution
After launching the Web browser page of Ms. Pac-Man(www.webpacman.com/Ms. Pac-Man.htm), ICE Pambush 4 can be run by moving to /src/ and executing
java main/MsPacInterface.

As in ICE Pambush 3, 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 one shown in the above video clip, where all objects are shown in the colors described above. The three figures below summarize from left to right the procedure for executing our controller.
Our controller should easily earn more than 5000 scores. However, if ICE Pambush 4 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) are as follows:

- **pinky** = -18210;       // = -18689 for the Microsoft Revenge Version
- **inky** = -16711714;    // = -16711681
- **sue** = -18361;        // = -18859
- **edible** = -14605858;  // = -14408449
- **pill1** = -2171170;    // = -2434305
- **pill4** = -2171170;    // = -2434305

Also in `GameParameter.java`:
- **brown** = -2189497;    // = -2453163
- **white** = -2171170;    // = -2434305

And `AutoSet.java`:
- **wall** = -18281;        // = -18774
- **wall2** = -14605858;    // = -12011777
- **wall3** = -2189497;     // = -2453163

Please try first to modify those parameters in red. After modifying those parameters, please recompile (JDK 1.6) the programs. Hope this helps!

3. **Controller**

The controller moves Ms. Pac-Man based on path costs. We adopted two versions of the Depth First Search (DFS) 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 ten rules (in decreasing priority) is fired.

**#Rule 1:** (Applied to Map1 only)

IF (distance(nearest_power_pill) \( \leq 5 \)) AND (3 \( \leq \) distance(nearest_ghost) \( \leq 10 \)) AND (distance(ghost_nearest_to_the_nearest_power_pill) \( \geq 7 \)),

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:** (Applied to Map1 only)

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) \( \leq 3 \)) OR (distance(ghost_nearest_to_the_nearest_power_pill) \( \leq 7 \))),

THEN move to the nearest power pill with DFS-B.

**#Rule 3:**

IF at least one power pill exists AND no edible ghost exists AND (distance(nearest_ghost) \( \leq 10 \)) AND (distance(nearest_power_pill) \( \leq 4 \)),

THEN move to the nearest power pill with DFS-B.

**#Rule 4:**

IF at least one power pill exists AND no edible ghost exists AND (distance(nearest_ghost) \( \leq 10 \)),

THEN move to the nearest power pill with DFS-A.

**#Rule 5:**

IF at least one edible ghost exists AND (distance(nearest_ghost) \( \leq 10 \)) AND (distance(nearest_edible_ghost) \( \leq 8 \)),

THEN move to the nearest edible ghost with DFS-A,

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) ≤ 10) AND
no pill# exists,
THEN move to the nearest pill with DFS-A,
where pill# indicate those pills not lying between a pair of two cross points whose connected path contains a power pill.

#Rule 7:
IF at least one pill exists AND
(distance(nearest_ghost) ≤ 10),
THEN move to the nearest pill # with DFS-A

#Rule 8:
IF at least one edible ghost exists AND
(distance(nearest_ghost) ≥ 11) AND (distance(nearest_edible_ghost) ≤ 12),
THEN move to the nearest edible ghost with DFS-B.

#Rule 9:
IF at least one pill exists AND (no pill# exists OR
((distance(nearest_ghost) ≥ 11) AND (distance(nearest_power_pill) ≥ 20) AND (distance(pill) ≤ 8))),
THEN move to the nearest pill with DFS-B

#Rule 10:
IF at least one pill exists AND
(distance(nearest_ghost) ≥ 11),
THEN move to the nearest pill# with DFS-B.

To find a path, DFS-A considers the distance cost, the ghost cost, and the corner cost while DFS-B considers only the distance cost, where cost definitions are given below. The former version of DFS is used in more critical situations, such as when a ghost is nearby, than the latter one. In addition, the depth of search $m$ is also different between these two versions. At a node at level $i$, where $i < m$, the current path will be expanded into four directions; i.e., north, east, south, or west; where the reverse direction and those towards the wall are excluded. In particular, the search depth of DFS-A and DFS-B 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:
= $[\text{distance}(X) + \text{distance}(X_{to\ target}) - \text{distance}(\text{target})]*1000$,
where X is the $i$th-level search corner or cross point from Ms. Pac-Man, $i = 1$ to 5 (DFS-A) or 1 to 10 (DFS-B), 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:
= $500,000/\text{distance}(Y_{to\ X})^2$,
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 a ghost resides:
= 2,000,000

#Ghost Cost III at a corner or cross point behind a ghost chasing Ms. Pac-Man on the same corridor:
= 500,000

# Corner Cost at each corner:
= 5,000

Reference