

# RAMP: A Rule-Based Agent for Ms. Pac-Man

Alan Fitzgerald, Peter Kemeraitis, and Clare Bates Congdon

May 23, 2008

**Abstract**—RAMP is a rule-based agent for playing Ms. Pac-Man according to the rules stipulated in the World Congress on Computational Intelligence Ms. Pac-Man Contest. Our architecture is still under development, but has thus far achieved a high score of 18,050 and typically scores over 10,000 in the current state of development.

## I. INTRODUCTION

MS. PAC-MAN is a classic arcade game originally released in 1981. In the game, the player controls the Ms. Pac-Man character as she eats her way through the dots on each of a series of mazes and tries to avoid the four ghosts that wander through the maze and that kill her on contact. On each maze level, Ms. Pac-Man must eat all the dots on the maze in order to advance to the next level; points are scored for each dot eaten. There are four power pellets per level that temporarily disable the ghosts, making them edible by Ms. Pac-Man and a source for additional points in the game. There are also treats that move through each maze level; for each treat eaten, Ms. Pac-Man scores additional points. The goal of the game is to score as many points as possible.

The game is more challenging than the better known Pac-Man game in that the ghosts' actions are randomized. Thus, the game is different each time it is played, and (unlike Pac-Man), a series of successful moves cannot be memorized.

In this paper, we describe our implementation of an artificial agent to play Ms. Pac-Man. This is an entry in the Ms. Pac-Man Competition at the IEEE World Congress on Computational Intelligence (WCCI 2008). Our system is called RAMP, a Rule-based Agent for Ms. Pac-Man. RAMP is still under development, but at the time of writing has achieved a high score of 18,050 in the game; the highest level we have attained is Level 5.

This paper will describe the implementation of the RAMP system, including a brief overview of our algorithms, which are still under development. We will also discuss some of the lessons we have learned in developing our agent.

## II. IMPLEMENTATION OVERVIEW

We are working in the following environment:

- MacOSX, version 10.5 (Leopard)
- Java, version 1.6
- Safari, version 3.1

Alan Fitzgerald, Peter Kemeraitis, and Clare Bates Congdon are with the Department of Computer Science, University of Southern Maine, Portland, ME 04104 USA (email: {afitz, kemerait, clare}@cs.usm.maine.edu).

To run the Ms. Pac-Man game, we use the version at <http://www.webpacman.com/>, one of the options specified in the contest rules.

Our agent code uses the provided software kit for the competition as the starting point. This provides us with the basic structure of the “screen scrape” of the pixels from the game and analysis of the locations of some of the objects as well as the controller that tells Ms. Pac-Man which direction to move.

### A. Agent Architecture

RAMP is designed as a rule-based approach to playing Ms. Pac-Man, with the intention that the rule sets may be subject to an evolutionary algorithms learning approach. However, by contest time, we did not have time to do any learning, so the rule sets used are hand coded.

The RAMP architecture is implemented with layers for both conditions and actions.

1) *Conditions*: At the lowest level of the conditions are the pixels themselves, which are analyzed to identify the higher-level features, such as the location of Ms. Pac-Man, the location of each of the ghosts (possibly edible), the locations of dots on the board, the locations of power pellets, and the locations of any treats on the board. The lowest level corresponds to a refinement of the code provided for the contest, which has been refactored to be more precise in terms of object identification.

At this level, characteristics of the current screen scrape must also be extracted, so that we are able to determine which maze we are playing, whether we are in an “intermission” in between levels (and should not attempt to issue commands), and whether the game is over.

Once we have identified the maze that we are looking at, we are able to preload a second level of information about that level, such as the xy location of each of the dots and powerpills, and the locations of “tunnels” that connect the left side of the board directly to the right side of the board and provide a sort of a short cut in some situations.

The third level of the conditions is abstractions useful to determining actions, such as how many ghosts are wandering the board at the moment, how close they are to me, and whether they are edible. These high-level abstractions form the basis for the conditions of the rule set.

Our set of conditions includes:

- Level number
- Map number (there are four maps in total in the game; for example, Level 1 and Level 2 both use the first map)

- The number of ghosts that are considered “very close” to us
- The number of ghosts that are considered “close” to us
- The number of ghosts that are currently roaming the maze
- The number of power pellets left in the maze
- Whether or not the ghosts are edible
- The number of edible ghosts that are close enough to eat
- Whether or not the fruit is close enough to eat

2) *Actions*: The actions have two levels. At the lowest level, the agent must communicate which direction to move Ms. Pac-Man, which is one of the four directions of the arrow keys: Left, Up, Right, or Down. At a higher level of abstraction, the actions of the rules can be to graze on the dots in the immediate vicinity, to evade a nearby ghost, or to head for the nearest power pellet.

Our set of possible actions includes:

- Graze (eat nearby pills)
- Eat edible ghosts
- Evade the ghosts
- Run to a power pellet

### B. Example Rules

Using the high-level conditions and actions, we currently hand code rule sets, containing rules that say, for example, “if there are no ghosts nearby, graze” and “if there is an edible ghost nearby, eat it”. As mentioned above, we have designed our system to enable evolutionary learning of the rule sets, but have not yet implemented the learning. We also expect to benefit from learning ideal combinations of system parameters, such as what is “near” for a ghost that might eat us vs. what is “near” for a ghost that we might eat.

A successful rule set might contain the following instructions. (Note that there are many “don’t care” conditions, which are not mentioned here).

- If there are ghosts very near, run to a powerpill
- If there are edible ghosts near, try to eat them
- (default) Graze

## III. LESSONS LEARNED

Our first phase of development, refactoring and fine tuning the original contest code, led to high scores of approximately 7,000. In other words, the improvement of identification of salient features on the board contributed to our initial successes with playing the game.

Our second phase of development, adding the vocabulary of higher level conditions and actions, took much longer to accomplish (design and debug) than we had anticipated. However, this has ultimately been the right choice, as it has thus far increased the best score to 18,050. Furthermore, having moved to the higher level abstractions and the use of a rule set for agent decisions, we are in a position to increase our decision-making capabilities by adding additional abstractions to our vocabularies and thus enhancing the expressive power of the rule-based system.

In abstracting high-level features, we are occasionally thwarted by an approach that may be too reactive. The power pellets blink and might not be present in some screen scrapes. Ghosts can cross each others’ paths, and thus might temporarily “disappear” from a given screen scrape; ghosts also disappear when they use the tunnels. We have not yet done much work in attempting to retain knowledge in between screen scrapes.

In terms of achieving high scores, we have observed that in levels 1 and 2, behaviors such as herding the ghosts before eating a powerpill can help raise the score considerably by facilitating eating more ghosts. However, on levels 3-5, the “edible time” for the ghosts begins to shrink to the point where it is not as advantageous to place a large emphasis on eating the ghosts.

Finally, of course, the ability to evade ghosts that are trying to eat you is the most important skill to avoid dying. Our current agent has a tendency to allow itself to get cornered when ghosts come from two different directions. Avoiding cornering situations is perhaps the next best improvement we could add to our agent’s behaviors.

### ACKNOWLEDGEMENTS

We would like to thank Simon Lucas for organizing the competition and for providing the initial coding framework to work from. We would also like to thank the USM Department of Computer Science, the USM Research Computing Group, the USM College of Applied Science, Engineering, and Technology, the USM Provost’s Office, and the USM Student Senate for the funding that made our entry in this competition possible.