A Monte-Carlo Approach for Ghost Avoidance in the Ms. Pac-Man Game

Bruce K. B. Tong
Dept. of Electronic Engineering
City University of Hong Kong
kbtong@cityu.edu.hk

Chi Wan Sung
Dept. of Electronic Engineering
City University of Hong Kong
albert.sung@cityu.edu.hk

***Abstract*— Ms. Pac-Man is a challenging, classic arcade game that provides an interesting platform for Artificial Intelligence (AI) research. This paper reports the first Monte-Carlo approach to develop a ghost avoidance module of an intelligent agent that plays the game. Our experimental results show that the look-ahead ability of Monte-Carlo simulation often prevents Ms. Pac-Man being trapped by ghosts and reduces the chance of losing Ms. Pac-Man's life significantly. Our intelligent agent has achieved a high score of around 21,000. It is sometimes capable of clearing the first three stages and playing at the level of a novice human player.**

Keywords- Monte-Carlo simulation; Ms. Pac-Man; game; competition; computational intelligence

I. INTRODUCTION

Computer games are often used as test-beds for development of AI techniques. One of the reasons is that computer games share similar problems with the real world but with simpler rules and clearer objectives. In recent years, computer games have received much attention in AI and Computational Intelligence (CI) research. Some competitions have been held in various IEEE conferences, including the competitions of Ms. Pac-Man [1], Mario [2] and simulated car racing [3]. These competitions provide a level playing field to compare the performance of different CI techniques.

Pac-Man was a popular and classic arcade game developed by the Namco Company in 1980. Different variations of the games have been developed subsequently. The original Pac-Man is a one-player game where the human player controls Pac-Man to traverse a maze, avoid the four ghosts (non-player characters) and eat all the pills to clear a stage. The ghosts in the original version have deterministic actions and travel path, making it possible for players to find the optimal strategy to play the game. Ms. Pac-Man, a variation of the original game, adds random elements in the ghosts' movements. Based on the current game status, each ghost chooses its own path according to its own probability distribution. This variation avoids players hard-coding an optimal path and thus increases the fun and difficulty of the game.

Same as Pac-Man, the objective of Ms. Pac-Man in each stage is to eat all the pills throughout the maze in that stage. Totally, there are four mazes in the Ms. Pac-Man game. The first two stages utilize the first maze, the next three stages the second maze, the next three the third maze, and the next the fourth maze. Fig. 1 shows the screen capture of the first stage of the game. There are four ghosts, namely Blinky (red), Pinky (pink), Inky (light blue) and Sue (orange). The ghosts are movable non-player characters. They live in the cage (center of the maze) initially and go out one by one after the game has started. They aim to shorten the distance between Ms. Pac-Man and themselves, but each with different distribution of probability of random move.

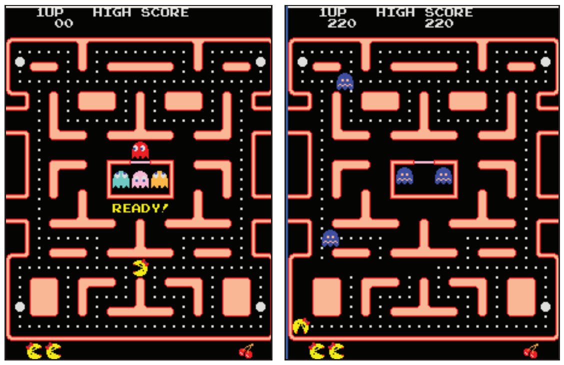


Figure 1. The first maze of Ms. Pac-Man (left) and the edible ghosts (right)

The number of pills is different between mazes. The first maze contains 220 pills (small white dot) and 4 power pills (big white dot). Both kinds of pills are non-movable objects. When a pill is eaten, the player receives 10 points. Once Ms. Pac-Man eats a power pill, the ghosts are changed to edible state (dark blue) temporarily as shown in Fig. 1. The power pills give Ms. Pac-Man the ability to capture the edible ghosts, allowing a player to get many bonus points. Eating the power pill gets 50 points, the first ghost 200, the second consecutive ghost 400, the third consecutive ghost 800 and the fourth consecutive ghost 1600.

Bonus fruits occasionally appear in the maze and move throughout the game randomly. The score of eating bonus fruit ranges from 100 to 5,000, depending on which stage the player is playing. At the beginning of the game, there are three lives for Ms. Pac-Man. An extra life will be given for the first 10,000 points the player has earned.

This paper describes an initial Monte-Carlo approach for ghost avoidance in the Ms. Pac-Man game. The rest of this paper is organized as follows: in Section II we review the related research in Pac-Man and Ms. Pac-Man, Section III describes our designed framework of agent, Section IV describes the Monte-Carlo method for ghost avoidance, Section V presents details and results of our experiments, and Section VI provides conclusions and discusses the future work.

II. Related Work

A. Pac-Man

Koza [4] and Rosca [5] are two of the earliest researchers who use Pac-Man as an example problem domain to study the effectiveness of genetic programming for task prioritization. Koza's work utilized a modified version of the game with a maze that replicated the first level of Ms. Pac-Man and using different score values for the items. According to Szita and Lorincz [6], the score reported on Koza's implementation would have been equivalent to approximately 5,000 points which equivalent to human novice level in their Pac-Man version.

Bonet and Stauffer [7] proposed a reinforcement learning technique. They used a neural network and temporal difference learning (TDL) in a 10 x 10 Pac-Man-centred window. The mazes are simplified to have only one ghost and no power pills. They showed basic pill pursuit and ghost avoidance behaviors using complex learning tasks.

Gallagher and Ryan [8] firstly attempted to develop a rule-based evolutionary Pac-Man agent which based on a simple finite-state machine and parameterized rules to control the movement of Pac-Man. The parameters were evolved using Population-Based Incremental Learning (PBIL) algorithm. The learning process was run in a reduced version of Pac-Man with no power pills and one deterministic ghost. The results showed that it was able to achieve some degree of machine learning, although the complexity of the game has been reduced much.

Gallagher and Ledwich [9] proposed to use neuroevolution to learn to play Pac-Man based on the raw on-screen information. Their work utilized a modified version of the original game with no power pills and one non-deterministic ghost only. Although no evolved agent was able to clear a maze in their experiments, this work showed the potential of pure machine learning from very limited information.

Thompson, McMillan, Levine and Andrew [10] proposed a simple finite state machine with look-ahead strategy that plays Pac-Man. The agent look-ahead by deploying an A* searching algorithm to find the fastest path to the node containing pills. The optimized agent achieved 7,826 points in 50 games of Pac-Man.

B. Ms. Pac-Man

Lucas [11] described an initial approach to develop a Ms. Pac-man agent. His control algorithm used a neural network to evaluate the next move of Ms. Pac-Man. His agent utilized a feature vector composed of handcraft data consisting of the distances from Ms. Pac-Man to each ghost and edible ghost, to the nearest pill, to the nearest power pills and to the nearest junction. The best evolved agent ran in the full-scale Ms. Pac-Man game achieved 4,780 points in over 100 games, equivalent to a reasonable human novice.

Szita and Lorincz [6] proposed a cross-entropy optimization algorithm that learns to play Ms. Pac-Man. The best performing agent obtained an average score of 8,186, comparable to the performance of a set of five non-experienced human players played on the same version of the game, who averaged 8,064 points.

Wirth and Gallagher [12] described an influence-map model for producing a Ms. Pac-Man agent. The model captures the essentials of the game. The highest average score of the optimized agent is 6,848, which showed comparable performance to a novice human player.

Handa and Isozaki [13][14] proposed an evolutionary fuzzy system for playing Ms. Pac-Man. Their method employs fuzzy logic and the evolution of fuzzy rule parameters. In addition, reinforcement learning was proposed to learn the action in critical situation. The best evolved agent achieved 5,739 in 10 runs of Ms. Pac-Man game.

More recently RAMP [15], a rule-based Ms. Pac-Man agent developed by Fitzgerald, Kemeraitis and Congdon, won WCCI 2008, with a high score of 15,970 points and an average of 11,166.

Burrow and Lucas [16] compared the performance of learning to play Ms. Pac-Man using evolution and Temporal Difference Learning (TDL). The study was run on a modified version of Ms. Pac-Man and results showed that evolution of multi-layer perceptrons (MLP) performed better.

DeLooze and Viner [17] proposed using fuzzy Q-learning algorithm to train agent playing Ms. Pac-Man. The optimized agent averaged between 3,000 and 5,000 points.

Robles and Lucas [18] proposed a simple tree search method to evaluate the best path for Ms. Pac-Man. The tree was simplified to ignore the movements of ghosts and/or changes in ghost state. It achieved a high score of 15,640 and an average of 7,664 in CIG 2009 and was the runner-up in the competition. This work demonstrated the importance of look-ahead strategy of agent design.

ICE Pambush 2 [19] and 3 [20], developed by Thawonmas et al, has won the competition held in CEC 2009 and in CIG 2009 respectively. It achieved a maximum score of 24,640 and an average of 13,059 in CEC 2009 and a maximum of 30,010 and an average of 17,102 in CIG 2009. It is also a rule-based agent. It used A* algorithm to search for the pill with lowest cost which is composed mainly by the distances between Ms. Pac-Man and ghosts. The main reason that it achieved high score is it aggressively lures the ghosts to the power pills followed by ambushing them.

Different techniques have been applied on Pac-Man and Ms. Pac-Man. Except the scores reported from the competitions, it should be emphasized that many of the above quoted scores have been obtained on different version of Pac-Man and Ms. Pac-Man simulators. This section gives a rough idea of the relative performance of previous work only.

III. Framework of Agent

Our agent first captures the entire screen and looks for the game window. Once the game window has been found, it will capture the game screen (which is a rectangular area much smaller than the full screen) every 30 milliseconds periodically until the end of the game. After capturing the game screen, our agent will analyze and extract the game objects. Some of the objects are movable (e.g., Ms. Pac-Man, ghosts and fruits) while the others are unmovable (e.g., pills and power pills). Based on the extracted information and the history of the locations of the movable objects, the agent should decide the most suitable direction for Ms. Pac-Man's next move.

A. Game maze representation

We represent the game mazes using weighted graphs. Road intersections are represented by vertices. Two vertices are connected by an edge if the corresponding road intersections can reach each other without passing other intersections. The distance between these neighboring intersections is represented by the weight of the corresponding edge. This information about the topologies of mazes is preprocessed and stored in a file. At the beginning of the game, our agent loads the file into memory. It then computes the shortest distance between any pair of vertices using Dijkstra's algorithm, and the result is stored in memory. The shortest distance between any two positions is computed on-demand. Our algorithm is the same as that in [13]. We present it as follows:

We denote the two given points by p_a and p_b. In the mazes of the Ms Pac-Man game, a point p_i that does not lie on road intersections must be connected to two and only two intersections. We name them v_{i1}, and v_{i2}. There are four cases we need to consider:

- Case 1: p_a and p_b are both vertices
- Case 2: p_a is a vertex and p_b is not
- Case 3: p_b is a vertex and p_a is not
- Case 4: Neither p_a nor p_b is a vertex

For case 1, the result can be obtained directly as it has been computed by the Dijkstra's algorithm $Dijkstra(p_a, p_b)$ at the beginning of the game. For case 2, the shortest distance is $min(Dijkstra(p_a, v_{b1}) + distance(v_{b1}, p_b), Dijkstra(p_a, v_{b2}) + distance(v_{b2}, p_b))$ where $distance(a, b)$ is the straight distance between the two points a and b that lie on the same edge. Case 3 can be handled in the same way as in case 2. For case 4, the shortest distance is $min(distance(p_a, v_{a1}) + Dijkstra(v_{a1}, v_{b1}) + distance(v_{b1}, p_b), distance(p_a, v_{a1}) + Dijkstra(v_{a1}, v_{b2}) + distance(v_{b2}, p_b), distance(p_a, v_{a2}) + Dijkstra(v_{a2}, v_{b1}) + distance(v_{b1}, p_b), distance(p_a, v_{a2}) + Dijkstra(v_{a2}, v_{b2}) + distance(v_{b2}, p_b))$.

The shortest direction from one point to another can also be obtained simultaneously. Once the distance and direction have been computed, the results will be cached in memory. Later when the agent needs the information again, it can be retrieved without further calculations.

B. Game object recognition

We divide each maze into 28 × 30 square cells, each of which contains 8 × 8 pixels. There are two kinds of cells: passable or impassable. The walls and blocked area are examples of impassable cells. As these cells cannot be entered by Ms. Pac-Man, ghosts nor fruits, their color pixels always remain unchanged throughout the game. To reduce computation time, we ignore all impassable cells in the game object recognition process.

For the passable cells, they either are empty cells or contain pill or power pill, and sometimes with partial movable objects (Ms. Pac-man, ghosts and/or fruits) staying over them. To speed up the recognition process, the bitboard idea from Chess programming [21] is used. Sixty four 32-bit random numbers ($r_{x,y}$) have been assigned to the 8 x 8 pixels in the cell initially. The hash value of a cell is obtained by the equation:

$$hash(pixel) = \sum_{x,y=0}^{7} r_{x,y} \oplus pixel(x, y)$$

The corresponding pseudo code of obtaining the hash value of a cell is shown in Fig. 2. Pixel is represented by a 2D array of 32-bit integers. The hash value is also a 32-bit number which can uniquely identify the different combinations of the 64 pixels in a cell if the initial random numbers are selected carefully.

For most cases, passable cells do not contain movable objects. The hash value is either equal to the hash value of an empty cell, pill cell or power pill cell. The worst case will be sixteen cells' hash values are different from these three. It is when Ms. Pac-Man, the four ghosts and a bonus fruit each of them occupies two cells, which involves totally sixteen cells.

```
function hash(int[][] pixel) {
    int h = 0;
    for (i = 0; i < 8; i++)
        for (j = 0; j < 8, j++)
            h += r[i][j] ⊕ pixel[i][j];
    return h;
}
```

Figure 2. The pseudo code of obtaining the hash value of a cell

In order to determine these movable objects, ICE Pambush 2's pixel color counting method [19] together with the history of movable objects is used. The color counting method is to count the number of colors in the cell. For example, if the number of red color is greater than a certain threshold, it will be considered as Blinky. The pixel color counting method is simple but not accurate enough in some cases since it does not consider the shape of object but color only. Wrong recognition may happen when there is a same color object (e.g. bonus fruit) appearing on the screen. A simple solution is to consider the history of the object as well. For example, if there is a red-type object but Blinky (and edible ghosts) is not moving nearby previously, which guarantees that the red-type object is not Blinky.

Since calculating the hash value of cells is relatively fast, it is able to identify the locations of movable objects quickly and the further recognition of these objects can be left for the counting method.

C. *Multi-mode framework*

The objective of the game is to achieve highest scores. In order to do so, one must eat pills and fruits as many as possible, avoid being captured by the ghosts and eat power pills in suitable time followed by capturing the edible ghosts as many as possible, at the same time.

As our agent uses the Graph structure to model the map, the distances between ghosts and Ms. Pac-Man are computed accurately. If the ghosts are far away, we say that Ms. Pac-Man is in safe situation; otherwise it is in dangerous situation. We further divide the safe situation into different modes, which have different short term objectives, as described below.

If there is an edible ghost near Ms. Pac-Man and no non-edible ghost near her, Ms. Pac-Man will enter the capture mode. In this mode, her objective is to chase the edible ghosts and capture them as many as possible.

If there is a power pill near Ms. Pac-Man, she enters the ambush mode. In this mode, Ms. Pac-Man waits at the corner which is close to the power pill. When the ghosts are close enough, she eats the power pill and enters the capture mode to capture the nearby edible ghosts. If the ghosts are not approaching, Ms. Pac-Man leaves this mode (and enters the pill mode described below so as to eat other pills instead of wasting time at the corner). Since capturing the edible ghosts provides significant bonus scores in the game, the ambush mode is the key part of obtaining high scores in the agent design.

For all other cases, Ms. Pac-Man enters the pill mode. In this mode, her objective is to eat the pills as many and fast as possible in order to get higher scores and clear the stage.

Currently we use greedy methods in all the modes under the safe situations. When Ms. Pac-Man is in the capture mode, she chases the nearest ghost using shortest path. When she is in the pill mode, she moves towards the nearest pill with maximum ghost distance. When Ms. Pac-Man is in dangerous situation, Monte-Carlo simulation is used, which will be described in Section IV.

Our multi-mode framework design allows us to easily add (or remove) mode modules for including new (or removing old) features. For example, we observed that our agent sometimes cannot clear the stage as there are some orphan pills on disjoined edges. It is hard for current agent to plan a path to eat the pills (and at the same time avoid being captured by ghosts). In this case, we may add a new mode to handle this situation. If the number of remaining pills is less than a threshold value, Ms. Pac-Man enters the endgame mode. And we can design an algorithm to tackle the problem specifically.

D. *Mode transition*

Simply with the above multiple modes, we observe that the behavior of our agent is strange especially when she leaves a mode and enters another mode. The reason is that the short term objectives of different mode usually have conflict with each other. For example, in the capture mode Ms. Pac-Man chases the edible ghosts and may instruct her to move far away from a set of pills. But when the edible ghosts run too far away, Ms. Pac-Man may enter the pill mode and go in reverse direction (towards the set of pills) immediately. After several time units, one edible ghost may go back and is approaching Ms. Pac-Man (edible ghost's direction is purely random). At that moment, Ms. Pac-Man enters capture mode again and reverses her direction immediately. The above scenario is summarized in Fig. 3.

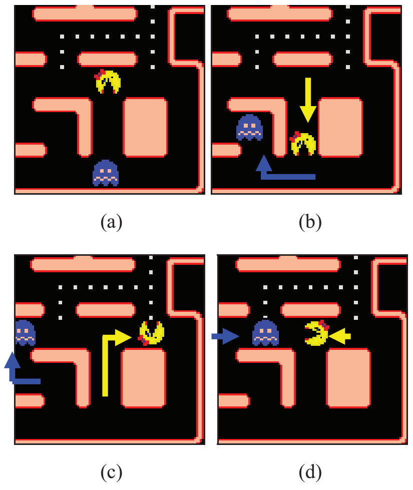


Figure 3. The scenario of mode conflicts: (a) Ms. Pac-Man enters the capture mode to chase the edible ghost (b) Ms. Pac-Man enters the pill mode when edible ghosts is too far away (c) the edible ghost approachs Ms. Pac-Man (due to random move) (d) Ms. Pac-Man enters the capture mode again to chase the edible ghost

The above problem cannot be adequately handled by using hysteresis alone. A better solution is to add transition modes in between the designed modes, and with hysteresis between neighboring modes. A transition mode can be modeled as a weighted function of the two connected modes. For example, the capture mode wants to minimize the distance between Ms. Pac-Man and edible ghosts ($f_{capture} = min(\sum distance(pacman, edible\ ghost\ i)))$ while the pill mode wants to minimize the distance between Ms. Pac-Man and a set of pills ($f_{pill} = min(\sum distance(pacman, pill\ i)))$. The transition mode between the capture mode and the pill mode is defined as $f_{capture\_pill} = f_{capture} \times \alpha + f_{pill} \times (1 - \alpha)$ where α is the weighting parameter $\in [0, 1]$.

IV. MONTE-CARLO SIMULATION IN ADVANCE IN DANGEROUS STATE

The Monte-Carlo simulation module is called when Ms. Pac-Man is in the dangerous state. We define the dangerous state as the case where Ms. Pac-Man is surrounded by one or more ghosts and likely to lose her life immediately if she makes a wrong move. An example of the dangerous state is shown in Fig. 4. It is not easy to design a simple static method to find the safe path as the ghosts are not static objects. The ghosts do move according to Ms. Pac-Man's action together with a probabilistic random decision. It is reasonable to use

Monte-Carlo simulation which has the look-ahead capability to obtain the accurate survival probability of different moving decisions of Ms. Pac-Man.

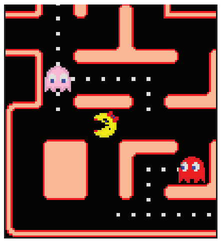


Figure 4. An example of dangerous situation

In our simulation module, we have made a number of assumptions and changes to the game rules so as to simplify the implementation and reduce the complexity. The assumptions and changes are listed as follows:

- The speeds of Ms. Pac-Man and the four ghosts are the same and are constant throughout the game.
- The coordinates of the Ms. Pac-Man and the four ghosts are integers. The coordinates reported from the recognition process have been rounded up to integer already.
- Ms. Pac-Man will decide her direction when she is in the intersecting point only. If we allow Ms. Pac-Man to change direction in any non-intersection point in the simulation, she will often perform random walk and the simulation result often do not reflect the truth value.
- Blinky moves towards Ms. Pac-Man's current coordinate by the shortest path with probability 0.9 when he is in the intersecting point. With probability 0.1, he selects a valid direction randomly.
- With probability 0.8 when Pinky is in the intersecting point, he goes ahead towards Ms. Pac-Man's next arrival vertex by the shortest path. But if he is already very close to the Ms. Pac-Man, he will move towards Ms. Pac-Man's current coordinate instead. With probability 0.2, he selects a valid direction randomly.
- With probability 0.7 when Inky is in the intersecting point, he follows Ms. Pac-Man's previous move by moving towards Ms. Pac-Man's previous vertex using the shortest path. But if he is already very close to the Ms. Pac-Man, he will move towards Ms. Pac-Man's current coordinate instead. With probability 0.3, he selects a valid direction randomly.
- With probability 0.5 when Sue is in the intersecting point, she moves towards Ms. Pac-Man's current coordinate by the shortest path. With probability 0.5, she selects a valid direction randomly.
- Ms. Pac-Man enters the energetic state after she has eaten a power pill. The time that Ms. Pac-Man will stay in this state is 30 units of cell travelling time. In other words, Ms. Pac-Man will resume to normal state after eating a power pill followed by travelling 30 cells.

The game tree of Ms. Pac-Man is relatively long and it is not feasible to take enough samples of simulation in real time. Our proposed Monte-Carlo method is a modified version. We perform simulation in advance and with shallow search instead of traditional full-depth search.

A. *Shallow search*

When the depth of the game tree is shortened, the leaf nodes represent the intermediate game states instead of ending game states. There is a problem in general situation but not in the dangerous state. The first reason is that it is not easy to decide a reasonable function to evaluate the leaf nodes as they are intermediate game states only. The evaluation of intermediate game states usually has conflicts with final game objectives. Maximizing the intermediate game state does not necessarily lead to a good ending game state and vice versa. The local optimum in shallow search is not necessarily the global optimum in full search.

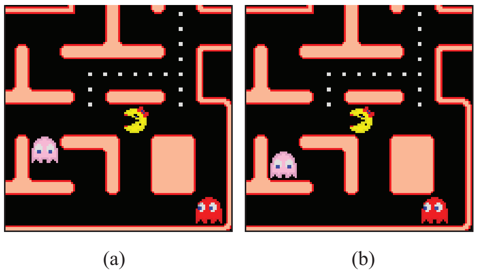
However in the dangerous state, when Ms. Pac-Man makes wrong decision she will die and that simulation will be stopped immediately. So these leaf nodes are not representing intermediate game states but ending game states. The evaluation of these ending game states is much easier and more accurate. Each simulation stops when Ms. Pac-Man dies or she remains alive after travelling a number of vertices. The simulation backward propagates the life status (alive or dead). The agent then selects the move with the maximum alive rate.

B. *Simulation in advance*

Just making shallow search does not completely solve the problem. As for all real-time games, the time constraint is very tight. For example, Ms. Pac-Man has to make a decision in the intersection of maze within about 10 milliseconds, otherwise she will move along the original direction and miss the intersection to make turn. The time is so short that it is not feasible to take enough samples in simulation.

To solve the problem, we simulate the next move from current situation in advance. The scenario is shown in Fig. 6. When Ms. Pac-Man is still in location (x, y), and we know that she is going along direction d, the next location (x', y') can be determined. From the moment that Ms. Pac-Man enters (x, y), simulation starts. The agent simulates the situation that Ms. Pac-Man is forced to enter (x', y') after one time unit (ghosts and fruits also update their locations accordingly), then she selects the subsequent moves randomly until the maximum distance or maximum number of vertices has been reached. The life status is backward propagated to (x', y') and another simulation from (x, y) is started again. When Ms. Pac-Man enters (x', y') in the playing game, simulations can be stopped. The simulation result is now retrieved and the move with maximum alive rate (e.g. direction d') can be decided. Now the current location of Ms. Pac-Man is (x', y') and current direction is d', the corresponding next location (x'', y'') can be determined. The next in advance simulation is going to be started immediately.

We observe that Ms. Pac-Man uses about 100 milliseconds to travel one cell in the game (the time is slower if the cell contains pill, and faster if Ms. Pac-Man turns around the corner). In other words, our agent can make use of the 100 milliseconds interval to simulate the next result. Experiments have shown that the time is now enough to take reasonable samples.



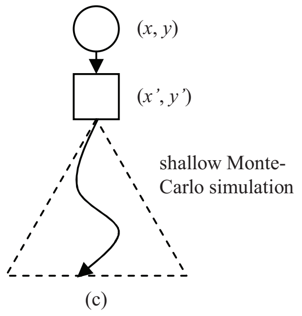


Figure 5. The scenario of simulation in advance: (a) starts simulation when Ms. Pac-Man in (x, y) (b) Ms. Pac-Man is forced to enter (x', y') in simulation (c) the subsequent moves is selected randomly and the life status is backward propagated to (x', y')

V. Performance Evaluation

We designed a strategy for ghost avoidance of Ms. Pac-Man agent by using Monte-Carlo simulation:

- *MC-Alive*: Simulation backward propagates the dead or alive condition. The depth of the shallow simulation is to visit at most five vertices of the game maze. The agent then selects a move with the maximum alive rate.

We designed another simple greedy strategy for ghost avoidance for comparison:

- *Greedy*: Selects a move to maximize the minimum distance between Ms. Pac-Man and ghosts. If there is a tie, maximize the second minimum distance and so on.

It should be emphasized that *MC-Alive* and *Greedy* use the same framework but invoke different modules for ghost avoidance in dangerous situation only. The dangerous state in the following experiments is defined as when the minimum distance between Ms. Pac-Man and the nearest ghost is less than 7 cells or the minimum distance between Ms. Pac-Man and the second nearest ghost is less than 10 cells. If it is already in dangerous situation, the minimum distance threshold is extended to 9 and 12 respectively so as to create the hysteresis effect.

A. *Experiments in the real Ms. Pac-Man game*

The experiments were conducted in the real Ms. Pac-Man game (via screen capture mechanism) for 20 games for each strategy that ran on Windows XP Professional SP3 on Intel Core-i7 860 2.8GHz CPU with 4GB Memory. The screen capture mechanism is widely used in the competition [1]. The average game time of *MC-Alive* and *Greedy* per game is roughly 10 minutes and 5 minutes respectively.

TABLE I. The scores of different strategies for 25 games

Agent	Min	Max	Mean	Std Dev
Greedy	1,980	13,260	6,361	3,473.7
MC-Alive	2,890	22,670	12,872	5,397.6

Agent	The 25th Percentile	Median	The 75th Percentile
Greedy	3,610	5,590	10,320
MC-Alive	8,980	13,860	16,990

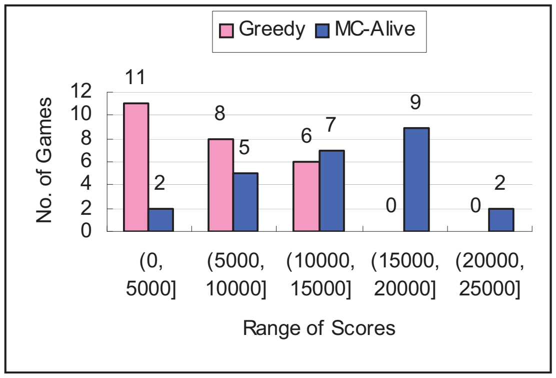


Figure 6. The score distrubution of the two strategies for 25 games.

The scores of the two strategies are listed in Table I. There is significant difference between the scores. *MC-Alive* achieved a high score of 22,670 and an average score of 12,872 in 25 games. *Greedy* achieved a maximum score of 13,260 and an average score of 6,361 only. The average score, 25th percentile, median and 75th percentile of *MC-Alive* are all higher and about double of those of *Greedy*. The look-ahead ability of Monte-Carlo simulation prevents Ms. Pac-Man being trapped by the ghosts. It successfully saves Ms. Pac-Man's life in dangerous situation, and allows the agent to play longer time and to achieve higher scores.

We also ran 25 games of ICE Pambush 3 [22], which is the only open source Ms. Pac-Man agent in the recent Ms. Pac-Man competitions, on the same system for comparison. The high score and the mean score are 36,860 and 18,440 respectively. Our scores are lower than that of ICE Pambush 3 by 38% and 30% respectively. But we have to reemphasize that our agent is currently focusing on ghost avoidance only and does not have any intelligent pills and ghosts hunting strategy but straight forward and greedy strategy only.

TABLE II. THE NUMBER OF STAGES CLEARED BY DIFFERENT STRATEGIES FOR 25 GAMES

Agent	Min	The 25th Percentile	Median	The 75th Percentile	Max
Greedy	0	0	0	1	1
MC-Alive	0	1	2	2	3

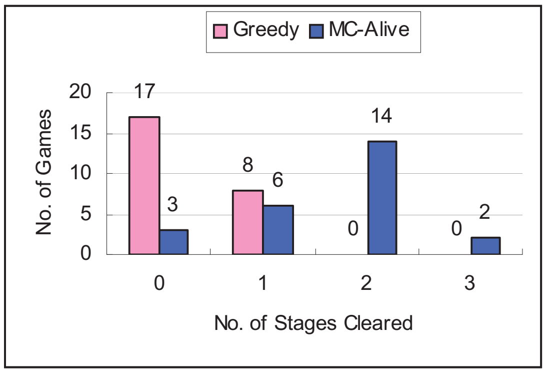


Figure 7. The distrubution of number of stages cleared by the two strategies for 25 games.

Besides the scores, we also compare the number of stages cleared by the two strategies. The results are listed in Table II. The results are consistent with that in Table I. *MC-Alive* is able to clear more stages than *Greedy* in general. 50% of the time, *Greedy* cannot clear the first stage. In other words, *Greedy* often loses all its three lives on stage 1. But *MC-Alive* is able to advance to stage 4 in its best run, and to the median of the number of cleared stages is three.

B. *Experiments in the dangerous situations*

In order to pay more attention to the dangerous situations, we conducted an individual set of experiments. Instead of running a whole game, different typical and dangerous situations were selected to test the ghost avoidance performance of the two strategies. In dangerous situations it seems to be a good move for Ms. Pac-Man in short term but often be a poor choice if she can look-ahead deeper in the long run. The correct move is sometimes a bad choice in short term but indeed a good move if we look-ahead evaluating it.

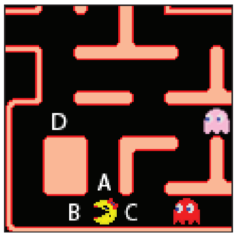


Figure 8. A selected example of dangerous situation

In the example of Fig. 8, Ms. Pac-Man has three choices, namely A, B and C. The correct move for Ms. Pac-Man should be moving towards A and escaping via D. *Greedy* advises Ms. Pac-Man to move to B since it maximizes the distances to Blinky and Pinky. But Ms. Pac-Man will be trapped in the corner easily. The right choice A although shortens the distance between Ms. Pac-Man and Pinky, *MC-Alive* is able to look-ahead and determine that Ms. Pac-Man can reach D via A and escape from the two ghosts.

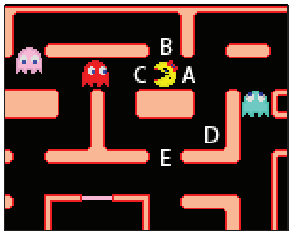


Figure 9. Another selected example of dangerous situation

Fig. 9 is another typical example. Ms. Pac-Man also has three choices, namely A, B and C. The correct move for Ms. Pac-Man should be moving towards A and escaping via D followed by E. *Greedy* advises Ms. Pac-Man to move to B since this move maximizes her distances to Blinky, Pinky and Inky. But Ms. Pac-Man will be trapped in the top long corridor easily. In contrast, *MC-Alive* selects A correctly since simulations report that B and C's alive rate are much lower than that of A.

VI. CONCLUSIONS

The aim of this paper is to study the effectiveness of applying Monte-Carlo simulation for ghost avoidance in the Ms. Pac-Man. This work is the first approach to decide the next move of Ms. Pac-Man in dangerous situation by Monte-Carlo method. The look-ahead ability of Monte-Carlo allows Ms. Pac-Man to find a safe path that avoids being captured by ghosts in a dynamic game environment. We propose a modified version of Monte-Carlo method. The shallow simulation simulates the situation in advance to overcome the tight time requirement in this real-time game. The result has shown that our method can reduce the chance of losing Ms. Pac-Man's life and raise the score significantly.

We are currently targeting to develop a high-scoring agent to participate in the Ms. Pac-Man competition. We plan to use CI techniques, instead of currently greedy method, to eat pills efficiently and to lure ghosts to power pills and ambush them effectively in order to achieve higher scores.

Monte-Carlo simulation is good at evaluating the average performance in short reasonable time, but weak in planning long term goals. For example, the result of using Monte-Carlo simulation to decide the path to eat the pills is unsatisfactory. We are considering Monte-Carlo Tree Search (MCTS), an extension of Monte-Carlo simulation applied to MinMax tree, to overcome this problem. MCTS is good at exploring promising paths in a tree and is successful in applying to Go [23][24] and Poker [25]. Further experiments of using MCTS in Ms. Pac-Man agent will be conducted.

REFERENCES

[1] S. M. Lucas, “Ms. Pac-Man competition,” SIGEVOlution, vol. 2, no. 4, pp. 37-38, 2007.

[2] J. Togelius, S. Karakovskiy and R. Baumgarten, “The 2009 Mario AI Competition,” IEEE Congress on Evolutionary Computation, pp. 1-8, 2010

[3] D. Loiacono, J. Togelius, P. L. Lanzi, L. Kinnaird-Heether, S. M. Lucas, M. Simmerson, D. Perez, R. G. Reynolds, Y. Saez “The WCCI 2008 Simulated Car Racing Competition,” IEEE Symposium on Computational Intelligence and Games, pp. 119-126, 2008

[4] J. R. Koza, Genetic Programming: on the Programing of Computers by Means of Natural Selection. MIT Press, 1992.

[5] J. P. Rosca, “Generality versus size in genetic programming.” Proceedings of the Genetic Programming 1996 Conference, pp. 381-387, MIT Press, 1996.

[6] Szita and A. Lorincz, “Learning to play using low-complexity rule-based policies: illustrations through Ms. Pac-Man,” Journal of Artificial Intelligence Research, vol. 30, no. 1, pp. 659-684, 2007.

[7] J. S. D. Bonet and C. P. Stauffer, “Learning to play Pac-Man using incremental reinforcement learning,” Proceedings of the Congress on Evolutionary Computation, 1999.

[8] M. Gallagher and A. Ryan, “Learning to play Pac-Man: an evolutionary, rule-based approach,” The 2003 Congress on Evolutionary Computation, vol. 4, pp. 2462-2469, 2003.

[9] M. Gallagher and M. Ledwich, “Evolving Pac-Man players: Can we learn from raw input?,” IEEE Symposium on Computational Intelligence and Games, pp. 282-287, 2007.

[10] T. Thompson, L. McMillan, J. Levine and A. Andrew, “An evaluation of the benefits of look-ahead in Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 310-315, 2008.

[11] S. M. Lucas, “Evolving a neural network location evaluator to play Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 203-210, 2005.

[12] N. Wirth and M. Gallagher, “An influence map model for playing Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 228-233, 2008.

[13] H. Handa and M. Isozaki, “Evolutionary fuzzy systems for generating better Ms. Pac-Man players,” IEEE International Conference on Fuzzy Systems, pp. 2182-2185, 2008.

[14] H. Handa, “Constitution of Ms. Pac-Man player with critical-situation learning mechanism,” Fourth International Workshop on Computational Intelligence & Applications, pp. 48-53, 2008

[15] A. Fitzgerald, P. Kemeraitis and C. B. Congdon, “RAMP: A rule-based agent for Ms. Pac-Man,” IEEE Congress on Evolutionary Computation, pp. 2646-2653, 2009.

[16] P. Burrow and S. M. Lucas, “Evolution versus temporal difference learning for learning to play Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 53-60, 2009.

[17] L. DeLooze and W. Viner, “Fuzzy Q-Learning in a Nondeterministic Environment: Developing an Intelligent Ms. Pac-Man Agent,” IEEE Symposium on Computational Intelligence and Games, pp. 162-169, 2009.

[18] D. Robles, S. M. Lucas, “A simple tree search method for playing Ms. Pac-Man,” IEEE Symposium on Computational Intelligence and Games, pp. 249-255, 2009.

[19] R. Thawonmas and H. Matsumoto, “Automatic controller of Ms. Pac-Man and its performance: Winner of the IEEE CEC 2009 software agent Ms. Pac-Man competition,” Proc. of Asia Simulation Conference 2009 (JSST 2009), 2009.

[20] H. Matsumoto, T. Ashida, Y. Ozasa, T. Maruyama and R. Thawonmas, “ICE Pambush 3,” Controller description paper, http://cswww.essex.ac.uk/staff/sml/pacman/cig2009/ICEPambush3/ICE%20Pambush%203.pdf (accessed 25/10/2010)

[21] A. L. Samuel, “Some studies in machine learning using the game of Checkers,” IBM Journal of Research and Development, vol. 3, issue 3, pp. 210-229, 1959.

[22] H. Matsumoto, T. Ashida, Y. Ozasa, T. Maruyama and R. Thawonmas, “ICE Pambush 3,” Controller source code, http://cswww.essex.ac.uk/staff/sml/pacman/cig2009/ICEPambush3/ICE%20Pambush%203.zip (accessed 25/10/2010)

[23] S. Gelly and D. Silver, “Achieving master level play in 9×9 computer go,” Proceedings of AAAI, pp. 1537–1540, 2008.

[24] C. S. Lee, M. H. Wang, G. Chaslot J. B. Hoock, A. Rimmel, O. Teytaud, S. R. Tsai, S. C. Hsu and T. P Hong, “The computational intelligence of MoGo revealed in Taiwan's Computer Go tournaments,” vol. 1, issue 1, pp. 73-89, 2009.

[25] G. V. d. Broeck, K. Driessens and J. Ramon, “Monte-Carlo Tree Search in Poker Using Expected Reward Distributions,” Proceedings of the 1st Asian Conference on Machine Learning: Advances in Machine Learning, pp. 367-381, 2009