Monte Carlo Tree Search for Collaboration Control of Ghosts in *Ms. Pac-Man*

Kien Quang Nguyen, *Student Member, IEEE*, and Ruck Thawonmas, *Senior Member, IEEE*

***Abstract*—In this paper, we present an application of Monte Carlo tree search (MCTS) to control ghosts in the game called *Ms. Pac-Man*. Our proposed ghost team consists of a ghost controlled by rules and three ghosts controlled individually by different MCTS. Given a limited time response, in order to increase the reliability of MCTS results, we introduce a mechanism for predicting Ms. Pac-Man's future movements and use this mechanism for simulating Ms. Pac-Man during Monte Carlo simulations. Our ghost team won the first Ms. Pac-Man Versus Ghost Team Competition at the 2011 IEEE Congress on Evolutionary Computation (CEC). Its performances for a variety of design choices are also shown and discussed.**

***Index Terms*—Ghosts, Monte Carlo, Monte Carlo tree search (MCTS), Pac-Man.**

I. Introduction

THE GAME OF *Ms. Pac-Man* is a video game that has simple rules but requires complex strategies for successful game play. The player must control Ms. Pac-Man to gain as many points as possible through eating edible objects, such as pills, in a maze without being killed by her four opponents, called ghosts. Although there is a considerable amount of existing work relating to automatic control of Ms. Pac-Man, there has been very limited work done in controlling the four ghosts in this game [1].

The main issue in ghost control is how to make them collaboratively catch Ms. Pac-Man. The common objective for all ghosts is to minimize the score earned by Ms. Pac-Man. Although they normally move with the same speed as Ms. Pac-Man, ghosts cannot reverse their directions. In addition, when ghosts are edible by Ms. Pac-Man, they can only move at half of their normal speed. Without effective collaboration control, it becomes nearly impossible for them to effectively trap and eat Ms. Pac-Man.

In this paper, in order to implement collaboration among ghosts, we adopt Monte Carlo tree search (MCTS) [2], in particular the UCB applied to trees (UCT) method [3], for controlling three ghosts (one MCTS per ghost), and a rule-based approach for controlling the remaining ghost. MCTS makes a decision based on tree search where nodes are evaluated through random simulations of future movements. As for UCT, it performs such simulations more often for paths whose nodes have higher rewards and are less often visited. This is done in order to keep balance between exploitation of paths containing higher rewarded nodes and exploration of paths containing less visited nodes in the tree.

In order to increase the reliability of such simulation results, given a limited response time, we propose another mechanism for predicting Ms. Pac-Man's movements and simulating such movements during Monte Carlo simulations based on its recent actual movements. Our ghost team ICE gUCT (also known as nqkien) won the first Ms Pac-Man Versus Ghost Team Competition [1], a game AI competition at the 2011 IEEE Congress on Evolutionary Computation (CEC), where the winner ghost team is the one that can best minimize the average score by all participant Ms. Pac-Man controller teams.

The contributions of this paper are as follows:

1) a detailed description of ICE gUCT, the 2011 IEEE CEC winner, that implements the following two salient mechanisms:
   a) a mechanism combining MCTS and a rule-based approach for controlling a team of multiple game AIs (ghosts) that can adapt to a variety of opponents;
   b) a mechanism for opponent (Ms. Pac-Man) modeling that maintains a high prediction accuracy up to a certain number of movements in advance.

II. Related Work

MCTS has gained in interest since its success in *Computer Go* [2]. Its state-of-the-art developments for *Computer Go* are summarized in [4]. Other recent applications of MCTS cover a variety of areas such as power-plant management [5], general game playing [6], turn-based games [7]–[13], and real-time games [14]–[17].

With respect to *Ms. Pac-Man*, Ikehata and Ito [16] applied the UCT method to controlling Ms. Pac-Man in the real game. However, their work has no mechanism for predicting the ghosts based on ghosts' recent movements. Samothrakis *et al.* [17] used $\max^n$ MCTS [18] to control both Ms. Pac-Man and a team of ghosts in a game simulator. Because Ms. Pac-Man's (and all ghosts') movements are considered in growing the $\max^n$ tree, their approach has a larger search space than our approach where separate MCTS is performed for each MCTS ghost.

Other existing work on ghost controllers includes the work of Yannakakis and Hallam [19] and that of Wittkamp *et al.* [20]. Yannakakis and Hallam's work discusses an online neuroevolution learning mechanism for controlling ghosts in order to meet

Manuscript received January 26, 2012; revised June 09, 2012 and July 19, 2012; accepted August 16, 2012. Date of publication September 19, 2012; date of current version March 13, 2013. This work was supported in part by the MEXT-Supported Program for the Strategic Research Foundation at Private Universities (2010–2014).

K. Q. Nguyen is with the Intelligent Computer Entertainment Laboratory, Ritsumeikan University, Kusatsu, Shiga 603-8577, Japan (e-mail: kien.n.quang@gmail.com).

R. Thawonmas is with the Department of Human and Computer Intelligence, Ritsumeikan University, Kusatsu, Shiga 603-8577, Japan (e-mail: ruck@ci.ritsumei.ac.jp).

Digital Object Identifier 10.1109/TCIAIG.2012.2214776

user satisfaction criteria proposed therein. The work in Wittkamp *et al.* is more related to our work. Rather than MCTS, it uses neuroevolution through augmenting topologies (NEAT) [21] for evolving a ghost through a simulated play whose timing is one time slice ahead. In such simulated plays, the movements of Pac-Man are simulated with Pac-Man in the actual game; in other words, a perfect Pac-Man model is used. Such information is not available in the Ms Pac-Man Versus Ghost Team Competition.

Recently, Lisy *et al.* [22] compared MCTS and the iterative deepening search algorithm (IDS) in a multiagent visibility based pursuit-evasion game and showed that IDS outperformed MCTS when both were used to control an evader agent. They stated that such a result was obtained because the evader agent has a limited number of options it can move. It might be worth comparing MCTS and IDS in the controlling of Ms. Pac-Man and/or ghosts, but this is beyond the scope of this paper.

III. *Ms. Pac-Man* Simulator

The *Ms. Pac-Man* simulator used in this research is the same as the one used in the Ms. Pac-Man Versus Ghost Team Competition at the 2011 IEEE CEC (version 1.0.4) [23]. It has four mazes (A, B, C, and D as shown in Fig. 1), the same as the original game, which are negotiated in that order. When maze D is cleared, the simulator goes back to maze A and continues the same sequence until the game is over. Each maze is modeled as a graph of connected grids. A grid contains a list of its neighbor grids. All distances between grid locations are stored in a lookup table in advance for use by both Ms. Pac-Man and ghost team controllers. The 2011 IEEE CEC competition officially allowed the use of multithreading, which eases the implementation of ghost control in a concurrent manner; this was unfortunately prohibited at the subsequent competition at the 2011 IEEE Conference on Computational Intelligence and Games (CIG) due to security concerns in the competition platform.

Ms. Pac-Man starts in maze A with three lives; an additional life is given at 10 000 points. Each pill and power pill eaten scores 10 points and 50 points, respectively. Four power pills reside in each maze. There are also four ghosts: Blinky (red), Pinky (pink), Sue (brown), and Inky (green). Their initial positions are inside a ghost cage. From this point, we call the ghost state *neutral*. When a ghost is out of the ghost cage, we call its ghost state *normal*. If a power pill is eaten by Ms. Pac-Man, all normal ghosts' directions will be reversed, and they can be eaten by Ms. Pac-Man during a certain period. We call such a ghost state *edible*. The more advanced the game level, the shorter the edible state lasts. The score for eating edible ghosts in succession starts at 200 and doubles each time a ghost is eaten until the end of the edible state. If eaten during the edible state, a ghost will become a pair of eyes, whose state we also call *neutral*, and returns to the ghost cage, where it will resume its normal state.

This simulator resembles the original *Ms. Pac-Man* game. However, there are still some important differences as follows.

- The speed of Ms. Pac-Man and that of the ghosts is always identical, except that edible ghosts move slower. In addition, their speed does not change even when they make a turn at a corner or run through a tunnel.
- Bonus fruits are omitted.

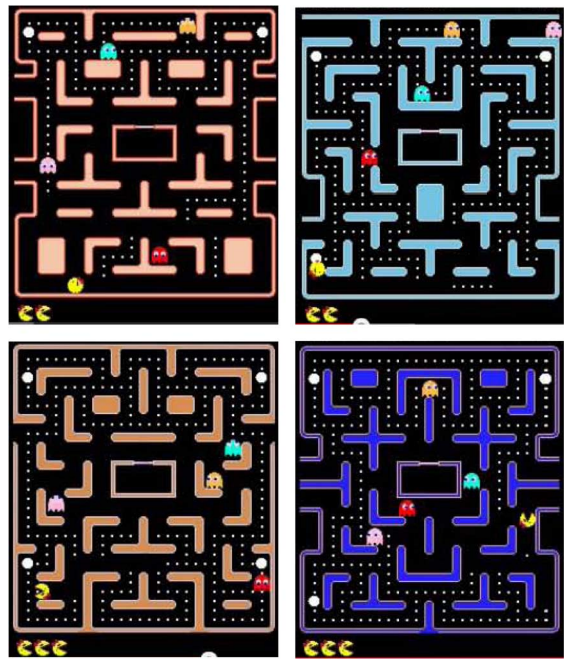


Fig. 1. Snapshots of mazes A, B, C, and D.

- Each maze is played once before shifting to the next one.
- Each level finishes after 3000 time steps, at which the scores of all remaining pills are awarded to Ms. Pac-Man.

The last point above was introduced to prevent the ghosts from constantly blocking a power pill in opposite directions, which spoils the game. All ghosts and Ms. Pac-Man move synchronously, one grid at a time. Every game cycle, the simulator sends the current game state to both controllers, waits for approximately 40 ms (response time), and then updates the new game state based on the directions received from the controllers.

IV. Overview of the Proposed Methodology

A. C Path and Time Slice

According to a game rule, a ghost is not allowed to reverse its direction. Therefore, with the exception of backwards, we can only control which direction it should go at a crosspoint. Henceforth, we refer to a corridor, connecting two adjacent crosspoints, as *C path*, borrowing the term from [16]. C paths are ghosts' primitive movement units. On the other hand, Ms. Pac-Man can change its direction at any grid so its primitive movement units are grids in the game maze. In the simulator, the distance between any two closest pills is four grids in the beginning of each maze, and, after analyzing all four game mazes, we find that the shortest distance between two adjacent crosspoints is three pills. As a result, each ghost has at least 480 ms to decide its new direction, unless a global reverse—a game event where all ghosts are forced to instantly reverse their directions—occurs, in which they have shorter movement-decision time.

Since a C path is a primitive movement unit, the time for a ghost to travel through a C path should also be considered as a ghost's primitive movement time unit. We refer to this amount of time as a *time slice*. Because the length of a C path is variable,

the length of a time slice also varies, depending on the position of a ghost of interest in the game maze.

B. Ghost Team Strategies

Our goal is to develop a strong ghost team that is able to trap and catch Ms. Pac-Man, controlled by any kind of Ms. Pac-Man controllers. To achieve this goal, we use the MCTS algorithm for controlling the directions of some ghosts. More specifically, we propose a ghost team controller where each of the following three ghosts—Pinky, Sue, and Inky—individually decides its direction with its tree and Blinky moves with its predefined rules discussed in Section IV-C.

We adopt the above combination of three MCTS ghosts and one rule-based ghost in order to balance the tradeoff between the reliability of MCTS ghosts and the complexity of rule-based ghosts. Namely, the more MCTS ghosts, the fewer playouts (number of visits to each tree's root node) can be performed for their trees, due to a constraint in the aforementioned response time, thus leading to less reliable resulting directions. However, if we used more rule-based ghosts, we would need to design more complex rules in order to coordinate their movements. In Section VIII, we compare the proposed ghost team controller ICE gUCT with a version that adopts four MCTS ghosts.

Given the limited response time, in order to increase the reliability of the MCTS ghosts, we propose a predictor for simulating Ms. Pac-Man's movements during Monte Carlo simulations. This predictor, whose detail is given in Section VI, uses in part the k-nearest neighbor (kNN) algorithm based on the information of Ms. Pac-Man's movements collected at every game cycle. In Section VIII, we also compare the performance of ICE gUCT and that of a version where the predictor is not used during Monte Carlo simulations.

C. Default Strategies

As our default strategies, we slightly modified a sample ghost team, Legacy, included in the 2011 IEEE CEC competition software. The default strategies of Blinky, Pinky, Inky, and Sue in the normal state move them toward the crosspoint that Ms. Pac-Man is heading to via their shortest paths based on the Dijkstra, Euclid, Manhattan, and Dijkstra distance metrics, respectively. In the edible state, only ghosts that are vulnerable to being eaten by Ms. Pac-Man will change their movement targets. In this case, such a ghost will move toward the surrounding grid that leads to the maximum increase in the distance, based on its distance metric, from Ms. Pac-Man.

V. Collaboration Control of Ghosts

By running Algorithm 1 at every game cycle, we control our MCTS ghosts in a concurrent manner, as shown in Fig. 2.

Algorithm 1: *getDirections*(*gameState*)

/* This algorithm returns the direction that MCTS ghost i at a crosspoint should go, given the current game state. Here, *gameState, nGhosts, dir*$[i]$, *rLvl,* n_{right}, n_{total}, and *nextSlice* denote the current game state, the number of the MCTS ghosts, the array containing the MCTS ghost directions, the reliability level of the Ms. Pac-Man's movement predictor *pacPred*, the number of correct predictions by *pacPred*, the number of total predictions by *pacPred*, and the game state at the beginning of the next time slice, respectively. */

```
1: if the game is restarted or a new level is reached then
2:    info ← new empty infoList
3: end if
4: curInfo ← Ms. Pac-Man's state information derived
   from gameState (cf. Section VI-A)
5: Append curInfo to info
6: for i ← 0 to nGhosts −1 do
7:    if ghost i is at the end of the current time slice then
      /* Stop construction of ghost i's tree used also
      by other MCTS ghosts during their Monte-Carlo
      simulations (cf. Section V-D) */
8:       MCTConstruction(MCT[i]).stop()
9:       MCT_prev [i] ← MCT[i]
10:      dir [i] ←chooseDirection(MCT_prev[i])
11:   end if
12: end for
13: rLvl ← (n_right/n_total)
14: pacPred ← new Predictor(info, rLvl) (cf.
    Section VI-C)
15: for i ← 0 to nGhosts −1 do
16:   nextSlice ← nextSliceGameState(i, gameState)
17:   if ghost i is (re)born at the ghost cage then
18:      root ← nextSlice.rootNode
19:      MCT [i] ←
            new MCT(i, root, pacPred, MCT_prev)
20:      MCT_initial [i] ← MCT[i]
         /* Start new thread */
21:      MCTConstruction(MCT[i]).start()
22:   end if
      /* The following condition is true when ghost i enters
      a new time slice or a globalreverse occurs */
23:   if nextSlice.rootNode ≠ MCT[i].rootNode then
         /* The following condition is true only when a
         global reverse occurs */
24:      if MCT_prev[i] ≠ MCT[i] then
25:         MCTConstruction(MCT[i]).stop()
26:         MCT_prev[i] ← MCT_initial[i]
27:      end if
28:      root ← nextSlice.rootNode
29:      MCT [i] ←
            new MCT(i, root, pacPred, MCT_prev)
30:      MCT_initial[i] ← MCT[i]
31:      MCTConstruction(MCT[i]).start()
32:   end if
33: end for
34: return dir
```

When an MCTS ghost enters a new time slice, we start constructing a new tree for it. Then, this tree continues to grow until either the ghost reaches the end of the current time slice or a global reverse occurs. At that moment, we stop the tree construction. For the former case, we determine the ghost's direction according to its tree (Algorithm 1). As an exception, when

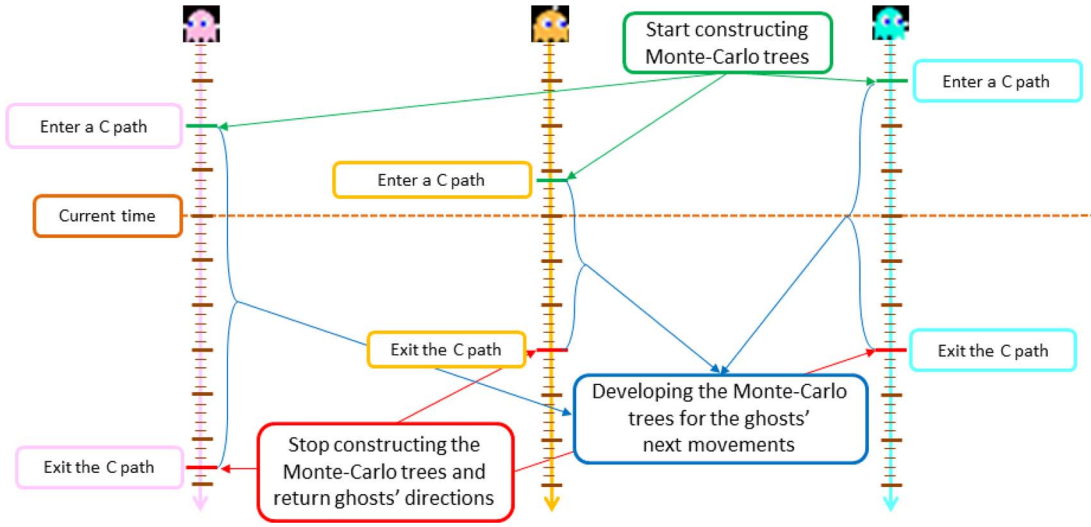


Fig. 2. An example of the timelines of the three MCTS ghosts: Pinky, Sue, and Inky.

the number of its MCTS playouts is still too small (less than threshold τ), we use our default strategies to decide its direction as shown in Algorithm 2.

Algorithm 2: *chooseDirection*($MCT[i]$)

/* This algorithm decides whether to use the direction from ghost i's tree ($MCT[i]$) or the default strategies to control ghost i, where τ is a predefined threshold. */

1: **if** $MCT[i].rootNode.visitedTimes < \tau$ **then**
2: **return** the direction to the first C path in the final path of $MCT[i]$ (cf. Section V-D)
3: **else**
4: **return** the direction decided by the default strategies of ghost i
5: **end if**

For the latter case, we start constructing a new tree after the global reverse. Although not shown in the figure, this process is then repeated. In the following sections, we describe our application of MCTS to ghost control in Section V-A, Monte Carlo simulations in Section V-B, node rewarding in Section V-C, and final path selection in Section V-D.

A. MCTS for Ghost Control

For an MCTS ghost, we construct a tree whose root node (initial node) is associated with the crosspoint that the ghost is going to visit next in the maze. According to the aforementioned game rule on ghosts' movements, it thus cannot turn back to the most recently visited crosspoint unless a global reverse occurs. Under this representation, a node represents a crosspoint, and a branch represents a C path. Following the standard procedure of MCTS (Algorithm 4), from the root node, we expand a path (Algorithm 5), start a Monte Carlo simulation, as described in Section V.B, and update each node's reward as described in Section V-C.

Algorithm 3: *nextSliceGameState*(i, *gameState*)

/* Get the game state of ghost i at the beginning of the next time slice */

1: $nextState \leftarrow gameState$
2: **while** $ghost[i].reachNextCrosspoint \neq TRUE$ **do**
/* Repeat the following three steps until the ghost reaches the next crosspoint */
3: Predict each ghost's position at the next game cycle (cf. Section V-B1)
4: Predict Ms. Pac-Man's position at the next game cycle (cf. Section V-B2)
5: Process the game events (e.g., those related to eating of pills, ghosts, or Ms. Pac-Man) based on the above predicted positions, and update *nextState*, accordingly.
6: **end while**
7: **return** *nextState*

Algorithm 4: *MCTConstruction* ($MCT[i]$)

/* Construct a tree for ghost i initialized with the root node */

1: **while** $MCT[i].rootNode.visitedTimes \leq maxVisitTime$ **do**
2: $path \leftarrow$ **expand**($MCT[i]$)
3: $simResult \leftarrow$ **simulation**($path$, i) (cf. Section V-B)
4: Update the reward for each visited node according to *time* and *score* in *simResult* (cf. Section V-C)
5: **end while**

Algorithm 5: *expand*($MCT[i]$)

/* Determine the best path according to UCB1 and expand it */

1: $path \leftarrow$ **getBestNodes**($MCT[i]$)

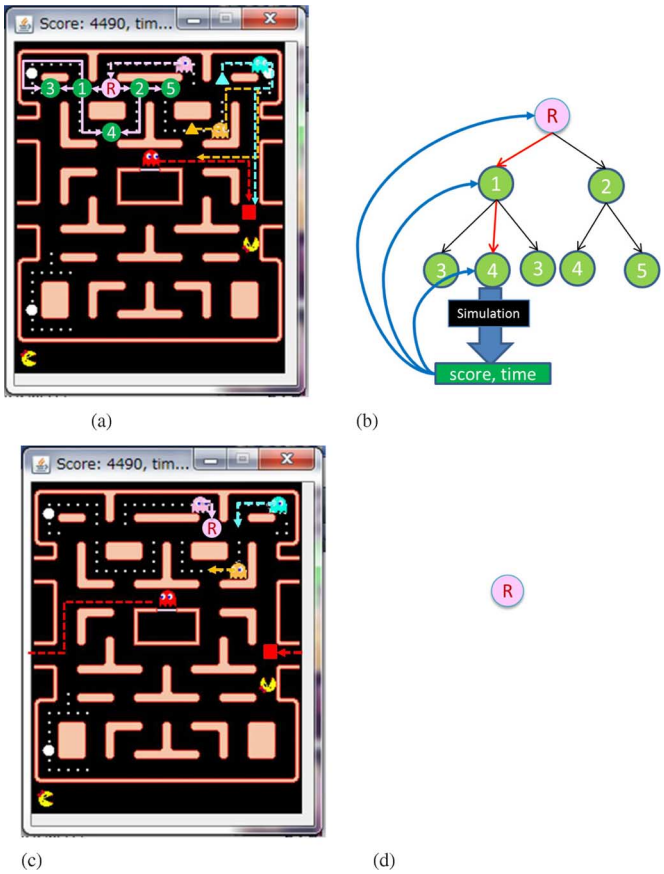


Fig. 3. Examples of game's maps and MCTS of Pinky (pink), where "R" represents the crosspoint that Pinky is approaching and corresponds to the root node in Pinky's tree; subsequent crosspoints are labeled in the map and shown in the tree accordingly. (a) Game's map. (b) An example of Pinky's MCTS for the left map. (c) Game's map after a global reverse occurs. (d) Pinky's tree newly initialized with the root node for the left map.

2: Randomly select a child node of *path.lastNode* and append it to *path*
3: **return** *path*

Traveling down from one node to another in its tree represents a simulated situation where an MCTS ghost of interest moves from the crosspoint corresponding to the former node to the crosspoint corresponding to the latter node.

Fig. 3 shows an example of the MCTS of Pinky at a given time slice. Fig. 3(a) shows typical ghosts' directions while Fig. 3(c) shows their directions right after a global reverse. In both figures, the current path of each ghost is shown by a dotted line whose destination is depicted by an arrow. Note that when a global reverse occurs, after reversing their directions, all MCTS ghosts end construction of their current trees and start constructing new ones.

During expansion of a path from the root node, for the latest nonleaf node in the path, we choose its child node that holds the largest UCB1 [24] value (Algorithm 6), among all child nodes, and append it to the path.

Algorithm 6: *getBestNodes(MCT[i])*

/* Select the best nodes from *MCT[i]*'s root node to a leaf node according to UCB1 */

1: *path* ← **new empty** *nodeList*
2: *curNode* ← *MCT[i].rootNode*
3: **while** *curNode is not a leaf node* ***do***
4: *Append curNode to path*
5: *curNode* ← $\underset{x \in \text{curNode.child}}{\arg\max}$ *UCB1(x)*
6: ***end while***
7: **return** *path*

Here, the child nodes of a node, say, node i, represent all adjacent crosspoints of the crosspoint corresponding to node i, excluding its parent, and the UCB1 value of node i is

$$UCB1(i) = \frac{X_i}{T_i} + C\sqrt{\frac{\ln T}{T_i + \epsilon}} \quad (1)$$

where X_i, C, T_i, and T denote the total reward of node i, a predefined constant, the number of times that node i was visited, and the number of times that its parent node was visited, respectively; ϵ is a small enough number ($\epsilon \approx 10^{-6}$). Because of this value of ϵ, the first time a child node is visited ($T_i = 0$), the node is assigned a large enough value so that unexplored moves are visited first.

B. Monte Carlo Simulation

In order to estimate the reward for each node residing in a selected path, from its root node to its leaf node, a Monte Carlo simulation is performed from the leaf node. We use the same simulation mechanism as that of the game simulator discussed in Section III. However, in our simulation mechanism, there is no global reverse, and we apply additional rules, discussed below, to the ghosts' and Ms. Pac-Man's movements. These movements are also used in Algorithm 3 for predicting the game state at the root node of an MCTS ghost of interest.

We stop a playout of interest **simulation**(*path, i*) in Algorithm 4, when one of the following events occurs.:

- Ms. Pac-Man is eaten;
- all pills and power pills are eaten;
- a certain amount of simulation cycles (called rollouts) is passed.

We stop each playout after a certain amount of rollouts in order to ensure a sufficient number of visits to the root node. In Section VIII, we discuss the performance of our ghost team in various numbers of rollouts.

1) Ghosts' Moving Rules: In a given MCTS playout of an MCTS ghost of interest, this ghost moves according to the path from the root node to the most recently appended leaf node. However, the paths of the other two MCTS ghosts, say, x and y, are those selected according to the mechanism in Section V-D using their previous trees, $MCT[x]_{\text{prev}}$ and $MCT[y]_{\text{prev}}$ in Algorithm 1, constructed during their previous time slices. For example, Fig. 3(a) shows all candidate paths for Pinky (an MCTS ghost of interest) and the selected paths of Sue and Inky, each starting from a triangle. After reaching its leaf node, each of these ghosts will randomly move toward a nonbackward direction once at a crosspoint.

For Blinky's movements during an MCTS playout, we propose two versions. In the first one, Blinky moves according to its rules in Section IV-C, as it does in the game, and in the

second one, adopted in ICE gUCT, Blinky randomly moves toward a nonbackward direction when it is at a crosspoint. In Section VIII, we compare the performances of these two versions.

2) Ms. Pac-Man's Moving Rules: For simulating Ms. Pac-Man, we adopt the following rules that simplify its movements.

- Ms. Pac-Man only makes a movement decision at a corner or a crosspoint. If the distance that it has moved is still shorter than a certain number of grids (called prediction cycles), its direction will be decided according to the proposed predictor in Section VI-C; otherwise, such a direction will become random. In Section VIII, we discuss the performance of our ghost team with the number of prediction cycles being varied.
- As an exception to the above rule, if there is a normal ghost ahead within a certain amount of grids and there is no power pill lying in between, Ms. Pac-Man will turn back suddenly.

C. Node Rewarding

After each playout, all nodes in the selected path, from the root to the leaf, of an MCTS ghost of interest will be equally rewarded according to the simulation result. We use two kinds of criteria to decide such a reward. The first is the inverse of the score earned by Ms. Pac-Man, and the second is the inverse of the simulation time. In addition, we consider that our MCTS ghosts should stay spread out from each other. They should also keep a close enough distance to Ms. Pac-Man in the normal state, in order to effectively make a pincer attack, while, in order to effectively avoid being eaten, keeping a safe distance from Ms. Pac-Man in the edible state. Therefore, we add a penalty to the score and another penalty to the time for each of the following two cases:

Case 1) there is another MCTS ghost in the first C path from the root node of the selected path;

Case 2) there are some MCTS ghosts that either do not stay close enough to Ms. Pac-Man when the ghost state is normal or stay far enough from Ms. Pac-Man when the ghost state is edible.

We also set the score for eating an edible ghost four times larger than the score for doing so in the game. This is done to penalize paths leading to a situation where an edible ghost will be killed by Ms. Pac-Man.

Let us assume that node i has been visited T_i times so far and that, at the jth playout, the corresponding simulation lasted for t_{ij}, during which Ms. Pac-Man earned s_{ij} points. In addition, $sp1$, $sp2$, $tp1$, and $tp2$ denote the score penalty for case 1, the score penalty for case 2, the time penalty for case 1, and the time penalty for case 2, respectively, where each penalty will be zero if the corresponding case does not hold and will have a predefined positive value otherwise. In the UCB1 formula (1), our definition of the total reward of node i, X_i, is

$$X_i = \alpha \cdot \min S_i \sum_{j}^{T_i} sr_{ij} + (1-\alpha) \cdot \min T_i \sum_{j}^{T_i} tr_{ij}$$

where α is a parameter whose value is within the range of $[0, 1]$; $\min S_i$ and $\min T_i$ denote the minimum score and the minimum simulation time among all T_i playouts, respectively; sr_{ij} and tr_{ij} denote the score reward and the time reward of node i at the jth playout, respectively, each given as

$$sr_{ij} = \frac{1}{s_{ij} + sp1 + sp2}$$
$$tr_{ij} = \frac{1}{t_{ij} + tp1 + tp2}.$$

In Section VIII, we compare ICE gUCT with a version that employs only the case 1 score and time penalties and another version that employs only the case 2 score and time penalties.

D. Final Path Selection

The direction of an MCTS ghost of interest, say, MCTS ghost x, must be decided based on its constructed tree $MCT_{\text{prev}}[x]$, when the ghost reaches a crosspoint. This decision is required not only in a game but also in a Monte Carlo simulation of another MCTS ghost. For the former case, we need only the direction leading to the first selected node (C path) while we need a sequence of directions leading to all selected nodes (C paths) that form a path for the latter case.

We select node i^*, among all candidate child nodes i, that has the highest average reward, i.e.,

$$i^* = \arg\max_i \frac{X_i}{T_i}.$$

In addition, for tie breaking, we select the node that has the highest T_i; if there are multiple nodes with that number, we randomly select one of them.

VI. Ms. Pac-Man's Movement Prediction

Given a current Ms. Pac-Man state, formed by the features discussed below, we predict Ms. Pac-Man's next movements based on its movement objectives in similar previous states. We simplify here and divide Ms. Pac-Man's movement objectives into two types: increasing or decreasing the distances between itself and certain objects, such as the nearest ghost.

A. Ms. Pac-Man States

Based on our experiences in developing a series of rule-based Ms. Pac-Man controllers for the Ms. Pac-Man competition [25], we use the following 10-dimensional vector $\mathbf{f}(t)$, for defining the state of Ms. Pac-Man P, at game cycle t. Let $d(x, y)$ and $pos(x, t)$ denote the Dijkstra distance between objects x and y and the position of object x at t, respectively. In addition, let $ng1$, $ng2$, $ng3$, npp, np, and nc denote the nearest ghost, the second nearest ghost, the ghost nearest to the nearest power pill, the nearest power pill, the nearest pill, and the nearest crosspoint, respectively. We define each element f_i of $\mathbf{f}(t)$ as follows:

- $f_1(t)$: the state of $ng1$;
- $f_2(t)$: the state of $ng2$;
- $f_3(t)$: the state of $ng3$;
- $f_4(t) = d(pos(P, t), pos(ng1, t))$;
- $f_5(t) = d(pos(P, t), pos(ng2, t))$;

- $f_6(t) = d(pos(P,t), pos(npp,t))$;
- $f_7(t) = d(pos(P,t), pos(np,t))$;
- $f_8(t) = d(pos(P,t), pos(nc,t))$;
- $f_9(t) = d(pos(P,t), pos(ng3,t))$;
- $f_{10}(t) = d(pos(npp,t), pos(ng3,t))$.

The first three features have three possible values: -1, 0, and 1 corresponding to *normal*, *neutral*, and *edible*, respectively. This is done such that the transition from the normal state to the edible state, or *vice versa*, leads to the highest difference in the value. The space spanned by vectors $\mathbf{f}$ is called *Ms. Pac-Man state space*.

B. Ms. Pac-Man Objectives

Focusing on relevant objects $ng1$, $ng2$, $ng3$, npp, np, and nc, we can express the objectives of Ms. Pac-Man's movement at t with respect to these objects by a 10-dimensional vector $\mathbf{o}(t)$ defined as

$$\begin{pmatrix} o_1(t) \\ o_2(t) \\ o_3(t) \\ o_4(t) \\ o_5(t) \\ o_6(t) \\ o_7(t) \\ o_8(t) \\ o_9(t) \\ o_{10}(t) \end{pmatrix} = \begin{pmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ d(pos(P,t+1), pos(ng1,t)) \\ d(pos(P,t+1), pos(ng2,t)) \\ d(pos(P,t+1), pos(npp,t)) \\ d(pos(P,t+1), pos(np,t)) \\ d(pos(P,t+1), pos(nc,t)) \\ d(pos(P,t+1), pos(ng3,t)) \\ f_{10}(t) \end{pmatrix} - \mathbf{f}(t). \quad (2)$$

Note that according to this definition, o_1, o_2, o_3, and o_{10} are always zero, because we only focus on an increase or decrease in the distance between Ms. Pac-Man and each of the above focused objects. For the other elements, their values will be negative if the objectives of Ms. Pac-Man's movement at t are to move toward the corresponding objects, given their positions fixed at t, and will be positive otherwise. The space spanned by vectors $\mathbf{o}$ is called *Ms. Pac-Man objective space*.

C. Predictor

Because $pos(P, t+1)$ is used in (2), only vectors $\mathbf{o}$ at previous game cycles are derivable. Henceforth, let t denote the current game cycle. As shown in Fig. 4, given $\mathbf{f}(t)$, we perform the following procedure for predicting which direction Ms. Pac-Man will take at t. First, we find previous vectors in the Ms. Pac-Man state space that are similar to $\mathbf{f}(t)$. Then, we use their counterpart vectors in the Ms. Pac-Man objective space to find an approximated vector of $\mathbf{o}(t)$, denoted as $\hat{\mathbf{o}}(t)$. In particular, $\hat{\mathbf{o}}(t)$ is the weight average of those counterpart vectors, the weight of which is the inverse of the distance to $\mathbf{f}(t)$. Next, we find the direction, among all possible directions, of Ms. Pac-Man at t whose corresponding vector in the Ms. Pac-Man objective space is nearest to $\hat{\mathbf{o}}(t)$. Eventually, using a probability derived from the reliability level of the predictor (*rLvl* in Algorithm 1), we select the final direction from a range between this direction and a randomly selected direction; the higher the reliability level, the more often the former direction will be chosen. This procedure is given in detail as follows.

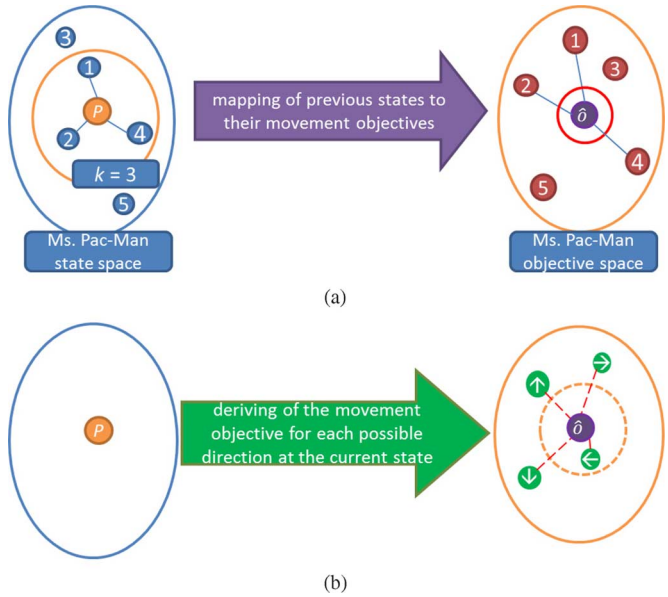


Fig. 4. Prediction of Ms. Pac-Man's next movement based on its movements objectives in similar previous states. (a) Predicting of the movement objectives at the current state from the current and previous game states of Ms. Pac-Man. (b) Predicting of the direction that should be taken by Ms. Pac-Man at the current state.

- Among R previous vectors in the Ms. Pac-Man state space, $\mathbf{f}(t-1), \mathbf{f}(t-2), \ldots, \mathbf{f}(t-R)$, find the nearest k vectors to $\mathbf{f}(t)$, denoted as $\mathbf{n}(t,1), \mathbf{n}(t,2), \ldots, \mathbf{n}(t,k)$.
- Compute $\hat{\mathbf{o}}(t)$ by

$$\hat{\mathbf{o}}(t) = \frac{\sum_{i=1}^{k} \frac{\mathbf{O}(t,i)}{d_H(\mathbf{f}(t), \mathbf{n}(t,i))}}{\sum_{i=1}^{k} \frac{1}{d_H(\mathbf{f}(t), \mathbf{n}(t,i))}}$$

  where $\mathbf{O}(t,i)$ denotes the counterpart vector in the Ms. Pac-Man objective space of $\mathbf{n}(t,i)$ and $d_H(\mathbf{f}(t), \mathbf{n}(t,i))$ represents the Manhattan distance between $\mathbf{f}(t)$ and $\mathbf{n}(t,i)$.
- For all possible directions to which Ms. Pac-Man can move at t, select the direction *selectedDir* that minimizes the Manhattan distance between $\hat{\mathbf{o}}(t)$ and $\mathbf{o}_{\text{dir}}(t)$, i.e.,

$$selectedDir \leftarrow \arg\min_{\text{dir}} d_H(\hat{\mathbf{o}}(t), \mathbf{o}_{\text{dir}}(t))$$

  where $\mathbf{o}_{\text{dir}}(t)$ denotes the resulting vector in the Ms. Pac-Man objective space, if we assume that Ms. Pac-Man decides to move to *dir* at t, by which the position of Ms. Pac-Man at $t+1$ and hence $\mathbf{o}_{\text{dir}}(t)$ can be derived.
- Select the final direction between *selectedDir* and a randomly selected *dir* with the probability of *rLvl* and $1-rLvl$, respectively.

If R in step 1 is too large, some earlier vectors might not fit for the current situation that Ms. Pac-Man is facing, leading to a low prediction accuracy. If R is too small, most of them will become the k nearest vectors to $\mathbf{f}(t)$ and be used in the computation of $\hat{\mathbf{o}}(t)$, regardless of their distances to $\mathbf{f}(t)$, which also leads

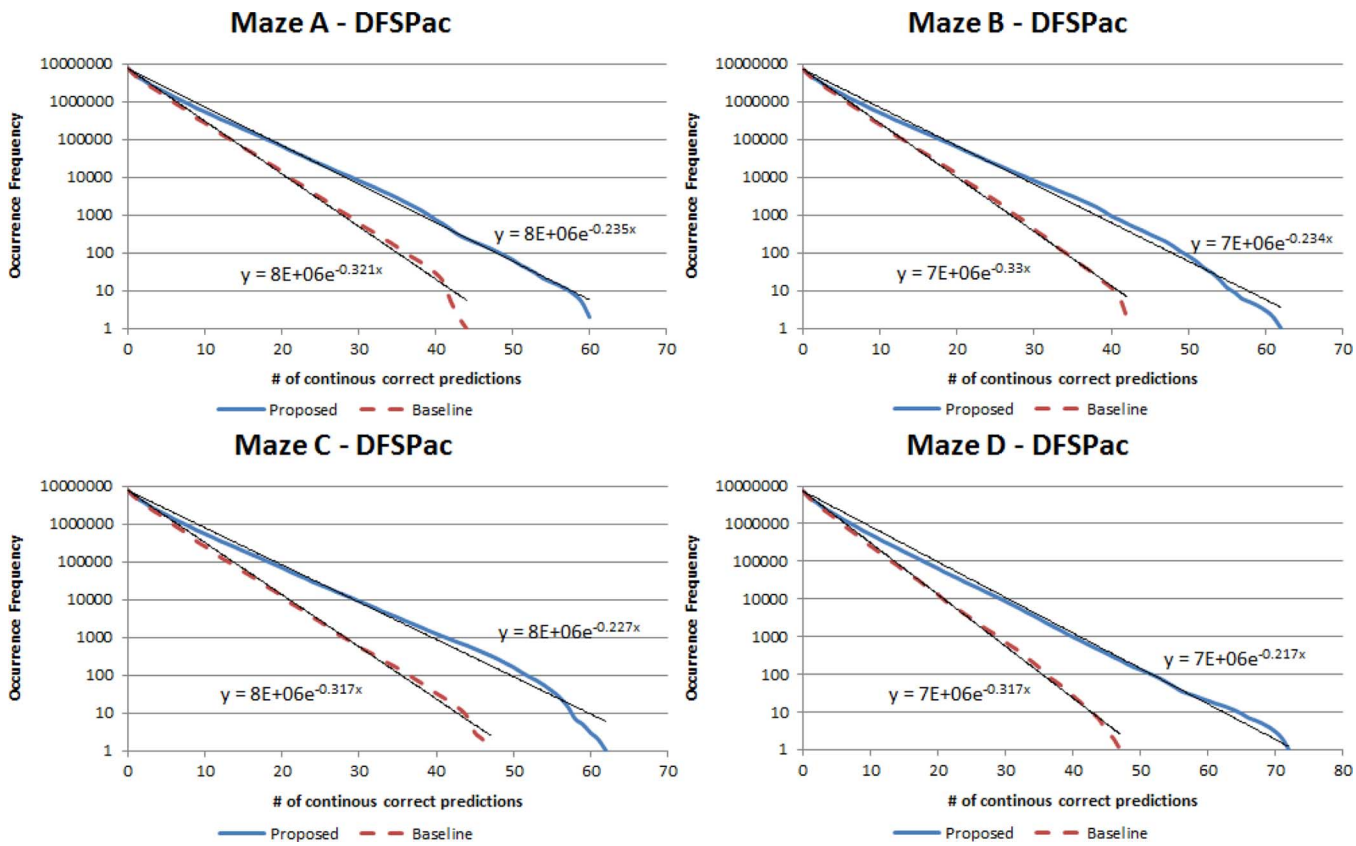


Fig. 5. Histograms showing the occurrence frequency in making continuously correct predictions of DFSPac's movements.

TABLE I
PERFORMANCE COMPARISON AMONG THE TOP THREE GHOSTS AT THE 2011 IEEE CEC COMPETITION

		Ghost Team Rank		
		1	2	3
Ms. Pac-Man Rank		ICE gUCT	BruteForce	emgallar
1	James	16436	24158	21805
2	emgallar	16208	22599	17938
3	MsAriadne	**19282**	**19031**	20076
4	mchammer	15283	15990	19425
5	BruteForce	12035	15028	12783
6	GhostBuster	10217	12673	13835
7	claygarrett	**10495**	**9586**	12676
8	Atif	**11282**	**5694**	13488
9	haimat	5829	8554	6878
10	essexgp	7163	8172	9164
11	RandomMsPacMan	1253	1795	1469

to a low prediction accuracy. In Section VIII, we examine the proposed ghost team controller with a variety of k and R.

VII. RESULTS AT THE 2011 IEEE CEC COMPETITION

As mentioned earlier, the proposed ghost team ICE gUCT came in first at the 2011 IEEE CEC competition. In this competition, there were eight ghost controllers and 11 Ms. Pac-Man controllers. As its summarized description, ICE gUCT used rule-based Blinky together with the other three MCTS ghosts, but moved Blinky randomly during Monte Carlo simulations. This version had the number of prediction cycles of 40, which is the maximum number of cycles during which the Ms. Pac-Man's movement predictor in Section VI is used. In addition, it had the number of rollouts of 400, considered both path (case 1) and distance (case 2) penalties for node rewarding, and used $k = 3$ and $R = 20$ in the predictor. The number of playouts was limited to 1000. These parameters were manually tuned based on the performances of ICE gUCT against Ms. Pac-Man controller entries, submitted during the test-play period, and against sample Ms. Pac-Man controllers, provided together with the competition software.

Table I compares the performance of ICE gUCT with the second place (BruteForce) and third place (emgallar) ghost controllers against all Ms. Pac-Man controllers. BruteForce used a set of hand-coded rules, while emgallar was based on ant colony optimization. This table shows that ICE gUCT outperformed BruteForce and emgallar against most Ms. Pac-Man controllers. It slightly underperformed BruteForce when they played against MsAriadne and claygarrett. There is one exception for the Ms. Pac-Man controller Atif, based on genetic programing, against which ICE gUCT lost significantly more points than BruteForce. However, because ICE gUCT still outperformed emgallar in this case, we consider that this was because Atif had serious weak points against the hand-coded BruteForce, rather than that ICE gUCT had weak points when it played against Atif.

VIII. EXPERIMENTS

We performed three experiments. The first experiment was aimed at analyzing the performance of the proposed mechanism for predicting Ms. Pac-Man's movements in terms of how

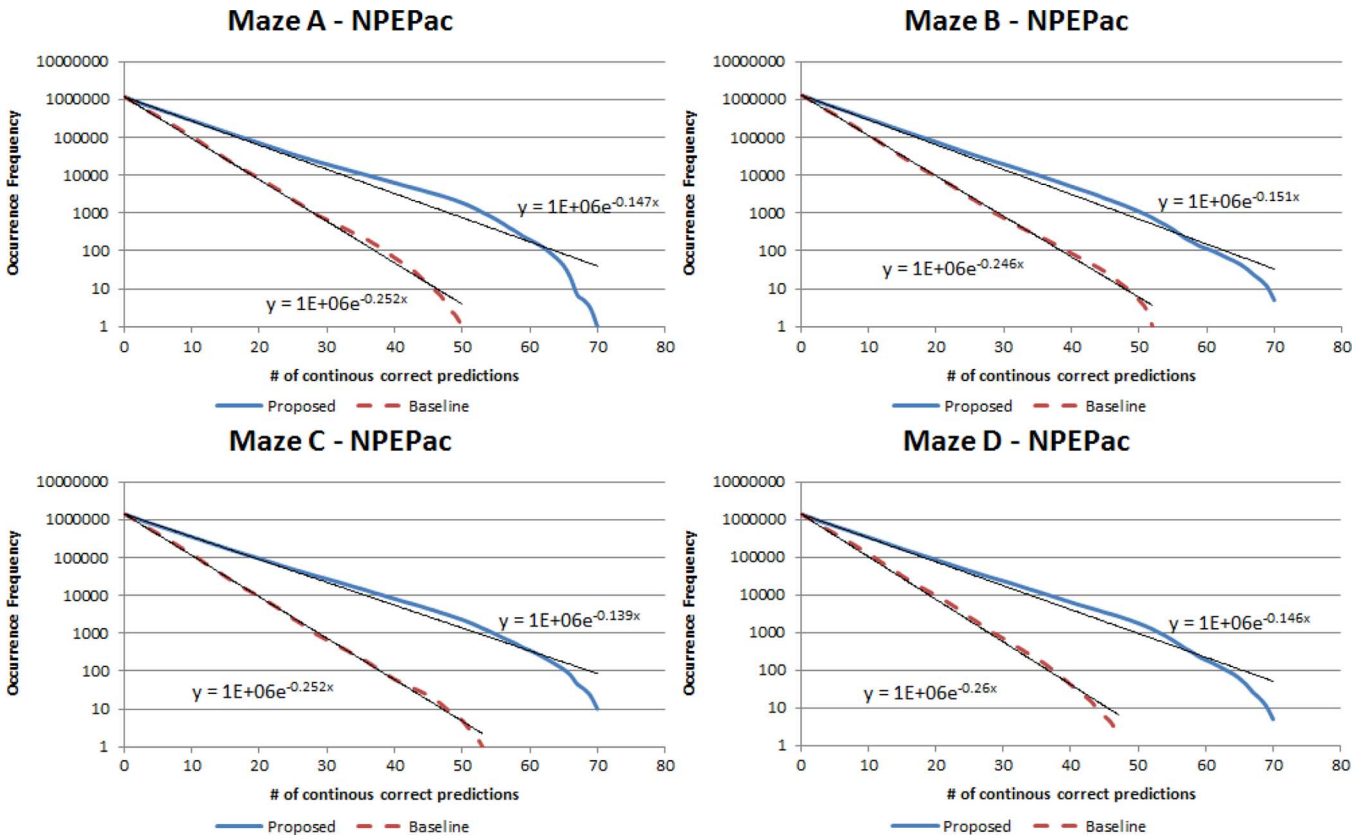


Fig. 6. Histograms showing the occurrence frequency in making continuously correct predictions of NPEPac's movements.

accurately it can predict Ms. Pac-Man's actual movements. The second experiment was to portray the characteristics of our MCTS implementation by examining how the number of rollouts and the number of prediction cycles affect the performance. In the third experiment, we compared ICE gUCT with a variety of versions that use different design choices.

We conducted these experiments on Intel(R) Core(TM) 2 Quad central processing unit (CPU) 2.83-GHz 4-GB RAM machines, with the ghosts being played against two types of Ms. Pac-Man: DFSPac that adopts the same rules as ICE Pambush 3 [26] and NPEPac that always moves to the nearest pill. For the first two experiments, we set C in (1), k, and R to 1, 3, and 20, respectively. In all experiments, we limited the number of playouts to 1000.

A. Performance of the Ms. Pac-Man Movement Predicting Mechanism

In this experiment, we evaluated the performance of our Ms. Pac-Man movement predicting mechanism on all four mazes over 100 games each. Note that, during Monte Carlo simulations, the more precisely we can simulate Ms. Pac-Man's movements, the faster (with a lower number of playouts) and more reliable the MCTS algorithm can decide its ghost's next direction. This idea is intuitive but was confirmed in previous work [12], [16]. Therefore, we here aimed at identifying in how many consecutive game cycles Ms. Pac-Man's actual movements can be correctly predicted.

We compared two predicting mechanisms. One is the proposed mechanism that predicts Ms. Pac-Man's movements according to Ms. Pac-Man's moving rules in Section V-B2. The other is a baseline mechanism that also uses the same moving rules but always replaces the results from the predictor in Section VI with random directions. We obtained the results in Figs. 5 and 6 through predicting the current Ms. Pac-Man's movement at a given game cycle, say t, with each mechanism based on each of Ms. Pac-Man' previous 200 state sets, each having R vectors $\mathbf{f}$. As mentioned above, we set R to 20 here, resulting in the set $\{\mathbf{f}(t-24), \mathbf{f}(t-20), \ldots, \mathbf{f}(t-4)\}$ (corresponding to four-cycle-ahead prediction), the set $\{\mathbf{f}(t-28), \mathbf{f}(t-24), \ldots, \mathbf{f}(t-8)\}$ (corresponding to eight-cycle-ahead prediction), ..., and the set $\{\mathbf{f}(t-820), \mathbf{f}(t-816), \ldots, \mathbf{f}(t-800)\}$ (corresponding to 800-cycle-ahead prediction). From Figs. 5 and 6, one can observe that the predictor, used in the proposed mechanism, can significantly increase the number of continuously correct predictions of the baseline mechanism. Because of its straightforward behaviors, as expected, NPEPac's movements can be more easily predicted than DFSPac. The probability in making x continuously correct predications for each mechanism is p^x shown in the corresponding regression equation Np^x, where p is the probability of making a single correct prediction (four cycles ahead) and N is the number of attempted predictions. For example, p of the proposed mechanism and the baseline mechanism for DFSPac in map A are 0.79 and 0.73, respectively, while for NPEPac, they are 0.86 and 0.78, respectively.

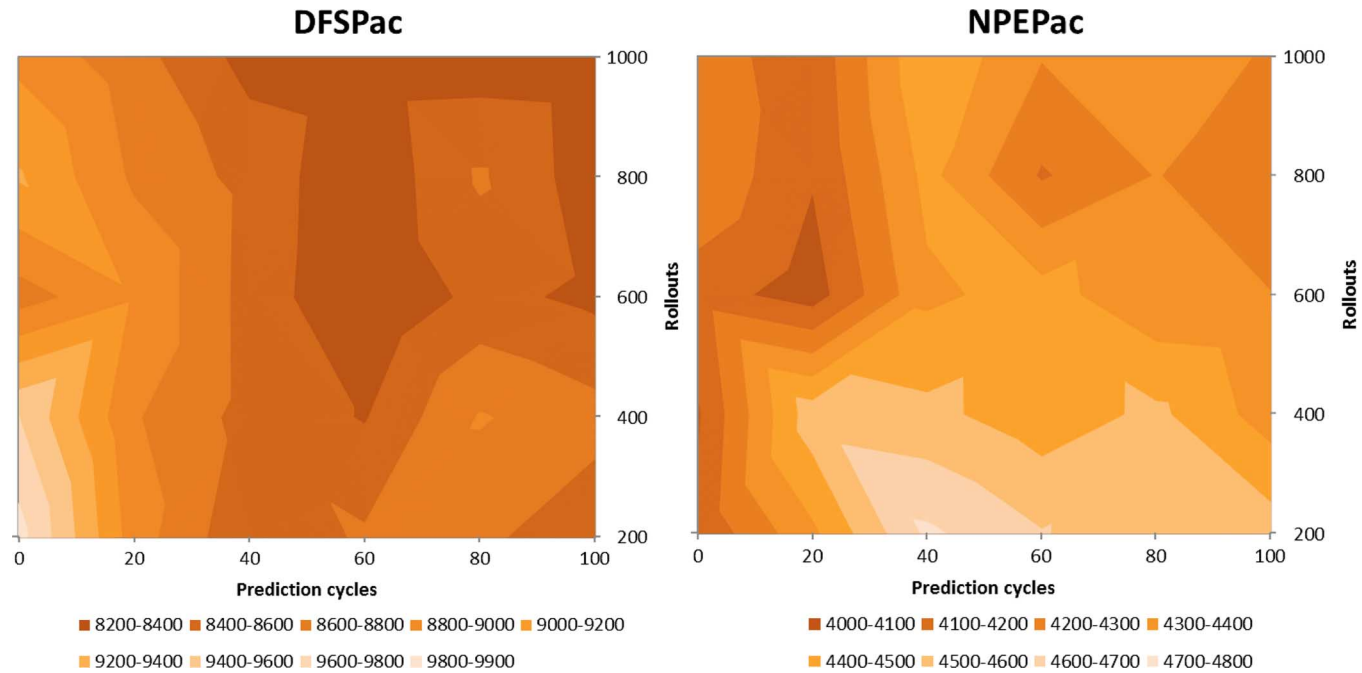


Fig. 7. The performance of our ghost team against DFSPac and NPEPac with various numbers of rollouts and prediction cycles, where adjacent color bands are different from each other with statistical significance.

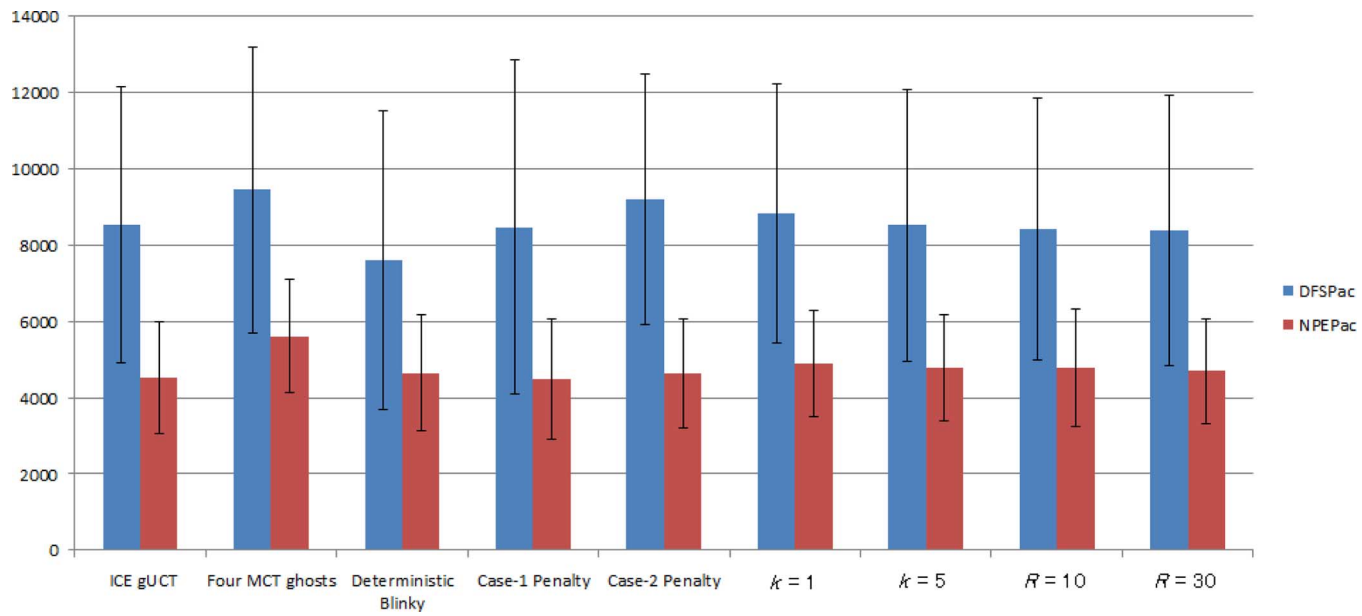


Fig. 8. Comparison of a number of variants of our ghost team, where the y-axis indicates the score obtained by Ms. Pac-Man.

B. Performance of the Ghost Team With Different Numbers of Rollouts and Prediction Cycles

In this experiment, we aimed at evaluating the performance of our ghost team with a variety in the numbers of rollouts and prediction cycles. The lower the score obtained by Ms. Pac-Man, the better performance the ghost team achieves. We progressively increased the number of rollouts from 200 to 1000, in increments of 200, and the number of prediction cycles from 0 to 100, in increments of 20. We note here that, given the limited response time of 40 ms, an increase in either parameter leads to a decrease in the number of playouts of Monte Carlo simulations, leading to a performance tradeoff. In order to obtain statistically significant differences among score ranges, we conducted 400 games for each set of parameters; we used the t-test with the significance level of $\alpha = 0.1$ for visualizing results in Fig. 7.

From the performance of our ghost team against DFSPac (Fig. 7), one can see that the best performance is achieved in the darkest color band residing around 60 and narrowly around 100 prediction cycles with the number of rollouts above 400 and 600, respectively. For NPEPac, the best performance is obtained in the darkest color band residing around 20 prediction cycles with the look-ahead cycle above 600. However, a large number of parameter sets can also give promising performances.

C. Performance of Various Ghost Team Versions

Here, we compared ICE gUCT, whose parameters were given in Section VII, with its variants (Fig. 8), each being run for 400 games. In the four-MCTS-ghosts version, all four ghosts move according to their MCTS results. The deterministic-Blinky version moves Blinky according to its rules during Monte Carlo simulations. In the case 1 penalty version, only the path score and time penalties are considered. The case 2 penalty version only considers the distance score and time penalties. For the remaining four versions in Fig. 8, the number of Ms. Pac-Man's game-state nearest vectors k or previous vectors R are changed accordingly. We conducted the t-test, with the significance level $\alpha = 0.05$, when we compared ICE gUCT with each variant.

From the results in Fig. 8, one can observe that the performance of ICE gUCT is superior to the version with all four ghosts being driven by MCTS. Their performance differences against both DFSPac and NPEPac are also statically significant, which validates our strategy to use a combination of one rule-based ghost and three MCTS ghosts as discussed in Section IV-B. The performance of ICE gUCT against DFSPac is inferior to the deterministic-Blinky version but outperforms the case 2 penalty version, with statistical significance. Changes in k and R do not lead to any performance difference against DFSPac, while against NPEPac, ICE gUCT outperforms three versions, i.e., $k = 1$, $k = 5$, and $R = 10$, with statistical significance.

In summary, ICE gUCT outperforms most of its variants. It is only inferior to the deterministic-Blinky version, which indicates that precise movements during Monte Carlo simulations of the rule-based ghost Blinky contributes to a better ghost team performance against a complex Ms. Pac-Man controller such as DFSPac. This finding will be explored by our ghost team in subsequent competitions.

IX. Conclusion and Future Work

We showed that the proposed approach—a combination of rule-based and MCTS methods—can be successfully used in a real-time game for collaboratively controlling a team of multiagents. The performance of this approach is better than that of the approach using only MCTS to control all agents. Our reason for this is that the proposed approach allows direct embedding of knowledge into the team's strategies. We used such knowledge in implementing our Blinky. In addition, our results show that exploitation of the same knowledge in Monte Carlo simulations gives a better performance, as shown in the performance of the deterministic-Blinky version.

We embedded another piece of knowledge in Ms. Pac-Man's movement rules and used these rules, together with the proposed predictor, for simulating Ms. Pac-Man's movements during Monte Carlo simulations. This predicting mechanism contributes to an increase in the performance of our ghost team. In addition, it enables our ghost team to effectively play against different types of Ms. Pac-Man controllers.

The approach presented in this paper should be readily applicable to other games that require real-time control of multiagents against their opponent(s). As our future work, we plan to expand our approach, such that determination of rule-based and MCTS-based agents can be done during games according to some criteria. In addition, we plan to apply online learning algorithms to learn the importance of each reward or penalty.

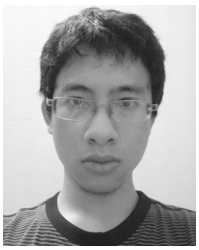
Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. They would also like to thank their lab members, in particular, the game AI group members, and Prof. F. Rinaldo, for their fruitful discussions.

References

[1] P. Rohlfshagen and S. M. Lucas, "Ms Pac-Man versus Ghost team CEC 2011 competition," in *Proc. IEEE Congr. Evol. Comput.*, 2011, pp. 70–77.
[2] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. 5th Int. Conf. Comput. Games*, 2006, pp. 72–83.
[3] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. 17th Eur. Conf. Mach. Learn.*, 2006, pp. 282–293.
[4] A. Rimmel, O. Teytaud, C.-S. Lee, S.-J. Yen, M.-H. Wang, and S.-R. Tsai, "Current frontiers in Computer Go," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 229–238, Dec. 2010.
[5] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *Proc. 5th Int. Conf. Learn. Intell. Optim.*, Italy, 2011, pp. 433–445.
[6] Y. Björnsson and H. Finnsson, "CadiaPlayer: A simulation-based general game player," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 1, pp. 4–15, Mar. 2009.
[7] R. J. Lorentz, "Amazons discover Monte-Carlo," in *Proc. Int. Conf. Comput. Games*, 2008, pp. 13–24.
[8] M. H. M. Winands, Y. Björnsson, and J.-T. Saito, "Monte Carlo tree search in *Lines of Action*," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 239–250, Dec. 2010.
[9] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in *Hex*," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 251–258, Dec. 2010.
[10] S.-J. Yen and J.-K. Yang, "Two-stage Monte Carlo tree search for *Connect6*," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 2, pp. 100–118, Jun. 2011.
[11] D. Whitehouse, E. J. Powley, and P. I. Cowling, "Determinization and information set Monte Carlo tree search for the card game Dou Di Zhu," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 87–94.
[12] D. Robles, P. Rohlfshagen, and S. M. Lucas, "Learning non-random moves for playing Othello: Improving Monte Carlo tree search," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 305–312.
[13] J. A. M. Nijssen and M. H. M. Winands, "Monte-Carlo tree search for the game of Scotland Yard," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 158–165.
[14] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game AI," in *Proc. 4th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2008, pp. 216–217.
[15] R.-K. Balla and A. Fern, "UCT for tactical assault planning in realtime strategy games," in *Proc. 21st Int. Joint Conf. Artif. Intell.*, 2009, pp. 40–45.
[16] N. Ikehata and T. Ito, "Monte-Carlo tree search in Ms. Pac-Man," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 39–46.
[17] S. Samothrakis, D. Robles, and S. Lucas, "Fast approximate max-n Monte Carlo tree search for *Ms. Pac-Man*," *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 2, pp. 142–154, Jun. 2011.
[18] N. Sturtevant, "An analysis of UCT in multi-player games," *Int. Comput. Games Assoc. J.*, vol. 31, no. 4, pp. 195–208, 2008.
[19] G. Yannakakis and J. Hallam, "A generic approach for generating interesting interactive Pac-Man opponents," in *Proc. IEEE Symp. Comput. Intell. Games*, 2005, pp. 94–101.
[20] M. Wittkamp, L. Barone, and P. Hingston, "Using NEAT for continuous adaptation and teamwork formation in Pacman," in *Proc. IEEE Symp. Comput. Intell. Games*, 2008, pp. 234–242.
[21] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[22] V. Lisý, B. Bošanský, and M. Pěchouček, "Anytime algorithms for multi-agent visibility-based pursuit-evasion games," in *Proc. Int. Conf. Autonom. Agents Multiagent Syst.*, 2012, vol. 3, pp. 1301–1302.

[23] CEC 2011 Ms. Pac-Man Competition: Submission Site [Online]. Available: http://cec11.pacman-vs-ghosts.net/
[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[25] Ms. Pac-Man Competition (Screen-Capture Version) [Online]. Available: http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html
[26] H. Matsumoto, T. Ashida, Y. Ozasa, T. Maruyama, and R. Thawonmas, "ICE Pambush 3," Team's Description, Aug. 30, 2009 [Online]. Available: http://cswww.essex.ac.uk/staff/sml/pacman/cig2009/ICEPambush3/



Kien Quang Nguyen (S'12) received the B.Eng. degree in human and computer intelligence from Ritsumeikan University, Kusatsu, Shiga, Japan, in March 2012, where he is currently working toward the M.Eng. degree in human information science, under the Japanese Government (MEXT) Scholarship.

His interests include machine learning, game theory, neural networks, and evolutionary algorithms.

Mr. Nguyen is the principal developer of the winning ghost team at the 2011 IEEE Congress on Evolutionary Computation (CEC) Ms. Pac-Man Versus Ghosts Competition. During his undergraduate study, he was a recipient of the Human Higher Education Development Support Project on ICT (HEDSPI) Scholarship.

Ruck Thawonmas (M'97–SM'99) received the B.Eng. degree in electrical engineering from Chulalongkorn University, Bangkok, Thailand, in 1987, the M.Eng. degree in information science from Ibaraki University, Ibaraki, Japan, in 1990, and the D.Eng. degree in information engineering from Tohoku University, Sendai, Japan, in 1994.

Before joining Ritsumeikan University, Kusatsu, Shiga, Japan, in April 2002, he had worked at various institutions: Hitachi, Ltd.; RIKEN; University of Aizu; and Kochi University of Technology. Since April 2004, he has been a Full Professor in the Department of Human and Computer Intelligence where he leads the Intelligent Computer Entertainment Laboratory. His research interests include game AI, automatic comic generation, and player-behavior analysis.

Dr. Thawonmas was a recipient of the Japanese Government Scholarship during 1987–1993. His laboratory has won a number of game AI competitions: the winning controllers at the 2009 IEEE Congress on Evolutionary Computation (CEC) and the 2009 IEEE Conference on Computational Intelligence and Games (CIG) Ms. Pac-Man Competitions (screen-capture version), the winning ghost team at the 2011 IEEE CEC Ms. Pac-Man Versus Ghosts Competition, and the winning human bot and judge bot at the 2011 BotPrize (at the 2011 IEEE CIG).